rexroth
A Bosch Company

IndraWorks 15VRS

Field Bus Libraries

Library
R911393275

Edition 05



Title IndraWorks
15VRS
Field Bus Libraries

Type of Documentation Library

Document Typecode DOK-IWORKS-FB*LIB**V15-LI05-EN-P

Internal File Reference RS-1d62254ca090d45fc0a802863e737572-5-en-US-3

Revision history Edition 5, 2026-01
Siehe tab. 1-1 "Revision history" on page 1

Copyright © Bosch Rexroth AG 2026
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 Control Functions PLC and Fieldbus, BeAn (TaDo/MePe)

Table of Contents

Page

1 About this documentation ... 1
1.1 Change record ... 1
1.2 Validity of the documentation ... 1
1.3 Libraries and target systems ... 2
1.4 Using safety instructions ... 3
1.4.1 Structure of the safety instructions ... 3
1.4.2 Explaining signal words and safety alert symbol ... 4
1.4.3 Symbols used ... 5
1.4.4 Explaining the signal alert symbol on the device ... 5
1.5 Required and supplementing documentation ... 5
1.6 Required and supplementing MLC documentations ... 6
1.7 Required and supplementing documentation MTX ... 12
1.7.1 Selection/compilation ... 12
1.7.2 Configuration ... 12
1.7.3 Commissioning ... 13
1.7.4 Operation ... 14
1.7.5 OEM Engineering ... 15
1.7.6 AddOns ... 15
1.8 Names and abbreviations ... 15
1.9 Customer feedback ... 16

2 Standard interfaces at function blocks ... 17
2.1 Motivation ... 17
2.2 Function block types ... 17
2.3 Block types and their interfaces ... 18
2.4 Inputs and outputs of state-controlled function blocks, final processing ... 19
2.5 Inputs and outputs of edge-controlled function blocks, final processing ... 22
2.6 Inputs and outputs of state-controlled function clocks, permanent processing ... 25
2.7 Inputs and outputs of edge-triggered function blocks, permanent processing ... 28

3 RIL_ProfibusDP_02.library ... 31
3.1 Introduction and overview ... 31
3.2 Data types ... 31
3.2.1 IL_BUSMASTER ... 31
3.2.2 Slave Diagnostic Data according to Profibus DP Standard ... 32
3.2.3 Bus master status word ... 34
3.2.4 Bus master error word ... 35
3.2.5 DP_SLAVELIST, array ... 35
3.2.6 DP_DEVICELIST, structure ... 36
3.2.7 DP_MASTERLIST, array ... 36
3.2.8 DP_MASTERINFO, structure ... 37
3.3 Selecting DP master ... 37
3.4 IL_DPMasterState ... 37

Page

3.5 IL_DPIdent ... 39
3.6 IL_DPV1Read ... 40
3.7 IL_DPV1Write ... 41
3.8 IL_DPReadDiag ... 43
3.9 IL_DPSycFr ... 45
3.10 IL_DPDeviceListGet ... 47
3.11 IL_DPBaudrateGet ... 48
3.12 IL_DPDevInfoGet ... 48
3.13 IL_DPPrjSlaveListGet ... 49
3.14 IL_DPSlaveDiagListGet ... 50
3.15 Error IDs ... 51
3.15.1 ErrorID / ERROR_CODE ... 51
3.15.2 ErrorIdent ... 52
3.15.3 FM_Error_Code ... 53
3.15.4 Error_Code_OB ... 54
3.15.5 Error_Code_FDL ... 54
3.15.6 Error_Code_DP ... 55

4 RIL_ProfibusDPSlave.library ... 57
4.1 Introduction and overview ... 57
4.2 IL_PBDPSlaveDPV1Polling ... 59
4.3 IL_PBDPSlaveDPV1Response ... 61
4.4 IL_PBDPSlaveDPV1GetWriteData ... 64
4.5 IL_BUSSLAVE ... 66
4.6 IL_FBUS_SLAVE_SERVICE ... 67
4.7 IL_FBUS_SLAVE_RESULT ... 67

5 RIL_ProfinetIO.library ... 69
5.1 Introduction and overview ... 69
5.2 Acyclic services (Acyclic Communication) ... 71
5.2.1 IL_PNIOReadRecord ... 71
5.2.2 IL_PNIOServiceRecord ... 74
5.2.3 IL_PNIOWriteRecord ... 79
5.3 Backing up and restoring (Backup_Restore) ... 82
5.3.1 IL_PNIODeviceBackup ... 82
5.3.2 IL_PNIODeviceRestore ... 86
5.4 Supporting the device replacement ... 89
5.4.1 IL_PNIOAssignName ... 89
5.4.2 IL_PNIOCheckTopology ... 91
5.4.3 IL_PNIODeviceReplacement ... 96
5.4.4 IL_PNIODeviceResetToDefault ... 98
5.4.5 IL_PNIODeviceSetIP ... 100
5.4.6 IL_PNIOGetLiveList ... 102
5.4.7 IL_PNIOGetLogMsg ... 104
5.4.8 IL_PNIOGetPortData ... 105

Page

5.4.9 IL_PNIOStoreTopology ... 108
5.4.10 IL_PNIORestoreTopology ... 109
5.5 Diagnostics ... 114
5.5.1 IL_PNIOControllerState ... 114
5.5.2 IL_PNIOControllerStateDetails ... 116
5.5.3 IL_PNIODiagnosis ... 118
5.5.4 IL_PNIOGetConfigDeviceNameList ... 122
5.5.5 IL_PNIOGetControllerList ... 124
5.5.6 IL_PNIOGetDiagDeviceNameList ... 125
5.5.7 IL_PNIORemoteDeviceState ... 126
5.5.8 IL_PNIORemoteDeviceStateDetails ... 128
5.6 I&M (Identification and Maintenance) (IM Data) ... 130
5.6.1 IL_PNIO_IM0 ... 130
5.6.2 IL_PNIO_IM1 ... 131
5.6.3 IL_PNIO_IM2 ... 131
5.6.4 IL_PNIO_IM3 ... 131
5.6.5 IL_PNIO_IM4 ... 131
5.6.6 IL_PNIO_IM_RWMODE ... 131
5.6.7 IL_PNIOReadWriteIMData ... 132
5.7 Communication interface (IO-Link) ... 137
5.7.1 IL_IOL_DEVICE_STATUS ... 137
5.7.2 IL_IOL_DEVICE_STATUS_DETAILS ... 137
5.7.3 IL_IOL_EVENT_QUALIFIER ... 137
5.7.4 IL_IOL_EVENT_QUALIFIER_INSTANCE ... 138
5.7.5 IL_IOL_EVENT_QUALIFIER_MODE ... 138
5.7.6 IL_IOL_EVENT_QUALIFIER_SOURCE ... 138
5.7.7 IL_IOL_EVENT_QUALIFIER_TYPE ... 138
5.7.8 IL_IOL_SERVICE ... 139
5.7.9 IL_IOL_BACKUP_MODE ... 139
5.7.10 IL_IOL_RESTORE_MODE ... 139
5.7.11 IL_PNIOIoLinkCall ... 139
5.7.12 IL_PNIOIoLinkGetDeviceAccessLock ... 145
5.7.13 IL_PNIOIoLinkIdent ... 150
5.7.14 IL_PNIOIoLinkIdentDetails ... 155
5.7.15 IL_PNIOIoLinkRemoteDeviceState ... 160
5.7.16 IL_PNIOIoLinkSetDeviceAccessLock ... 165
5.7.17 IL_PNIOIoLinkSystemCommand ... 170
5.7.18 IL_PNIOIoLinkBackupDevice ... 175
5.7.19 IL_PNIOIoLinkRestoreDevice ... 180
5.8 Data types ... 184
5.8.1 IL_BUSMASTER ... 184
5.8.2 IL_FBUS_COMMUNICATION_STATE ... 185
5.8.3 IL_FBUS_SLAVE_STATE ... 185
5.8.4 IL_PNIO_CONTROLLER_STATE ... 185
5.8.5 IL_PNIO_PORT_DATA ... 186
5.8.6 IL_PNIO_REMOTE_DEVICE_INFO ... 186

Page

5.8.7 IL_PNIO_REMOTE_DEVICE_STATE ... 187
5.8.8 IL_PNIO_TOPO_STATE ... 187
5.8.9 IL_PNIO_TOPO_STATUS ... 188
5.8.10 PNIO_CHANNELERROR ... 188
5.8.11 PNIO_DIAGINFO ... 189
5.8.12 PNIO_DIAGTYPE ... 189
5.8.13 PNIO_LIVELIST ... 189
5.9 PROFINETIO_TABLE 16#200 ... 190
5.9.1 General information ... 190
5.9.2 General Error Codes ... 191
5.9.3 Special Error Codes, Profinet I/O Controller ... 193
5.9.4 Special error codes, Profinet IO status ... 193
5.9.5 Special error codes, IO I/O link master ... 203
5.9.6 Special error codes, Profinet I/O I/O link device (application-related errors) ... 203
5.10 Example programs ... 204

6 RIL_ProfinetIODevice.library ... 205
6.1 Introduction and overview ... 205
6.2 Diagnostics (POUs/Diagnosis) ... 205
6.2.1 IL_PNIODeviceState ... 205
6.2.2 IL_PNIODeviceStateDetails ... 207
6.2.3 IL_PNIODeviceStateDetailsXMAC ... 208
6.3 Data types (DataTypes) ... 209
6.3.1 IL_PNIO_DEVICE_STATE ... 209
6.3.2 IL_PNIO_DEVICE_XMAC ... 211
6.3.3 IL_PNIO_DEVICE_XMAC_PORT ... 211
6.4 IL_BUSSLAVE ... 212
6.5 PROFINETIO_TABLE 16#200 ... 213
6.5.1 General information ... 213
6.5.2 General error codes ... 213
6.5.3 Special error codes, PROFINET I/O device ... 215

7 RIL_EtherNetIP.library ... 217
7.1 Introduction and overview ... 217
7.2 Acyclic services (Acyclic_Communication) ... 218
7.2.1 IL_ENIPGetAttributeSingle ... 218
7.2.2 IL_ENIPGetIdentityObjAllAttributes ... 220
7.2.3 IL_ENIPSendUnconnectedMessage ... 222
7.2.4 IL_ENIPSetAttributeSingle ... 224
7.3 Backing up and restoring (Backup Restore) ... 226
7.3.1 IL_ENIPAdapterBackup ... 226
7.3.2 IL_ENIPAdapterRestore ... 230
7.4 Diagnostics (Diagnosis) ... 232
7.4.1 IL_ENIPRemoteAdapterState ... 232
7.4.2 IL_ENIPRemoteAdapterStateDetails ... 233

Page

7.4.3 IL_ENIPScannerState ... 236
7.4.4 IL_ENIPScannerStateDetails ... 237
7.5 Data types (DataTypes) ... 239
7.5.1 IL_BUSMASTER ... 239
7.5.2 IL_ENIP_IDENTITY_ATTRIBUTES ... 239
7.5.3 IL_ENIP_REMOTE_ADAPTER_STATE_DETAILS ... 240
7.5.4 IL_ENIP_SCANNER_STATE ... 241
7.5.5 IL_FBUS_COMMUNICATION_STATE ... 242
7.5.6 IL_FBUS_SLAVE_STATE ... 242
7.5.7 Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151) ... 242

8 RIL_EtherNetIPAdapter.library ... 245
8.1 Introduction and overview ... 245
8.2 Diagnostics (Diagnosis) ... 245
8.2.1 IL_ENIPAdapterState ... 245
8.2.2 IL_ENIPAdapterStateDetails ... 247
8.3 Diagnostics (Diagnosis/EngineeringPortAdapter) ... 248
8.3.1 IL_Status ... 248
8.4 Data types (DataTypes) ... 250
8.4.1 IL_BUSSLAVE ... 250
8.4.2 IL_ENIP_ADAPTER_STATE ... 250
8.4.3 General error codes, EtherNet/IP adapter ... 251
8.4.4 Special error codes, EtherNet/IP adapter ... 253
8.4.5 CIP, general error codes ... 253
8.4.6 Advanced error codes for connection manager ... 255

9 RIL_Ethercat.library ... 257
9.1 Introduction and overview ... 257
9.2 Acyclic communication CoE (Acyclic Communication/CoE) ... 258
9.2.1 IL_ECATCoeReadSdo ... 258
9.2.2 IL_ECATCoeWriteSdo ... 261
9.3 Acyclic communication/SoE (Acyclic Communication/SoE) ... 263
9.3.1 IL_ECATSoeRead ... 263
9.3.2 IL_ECATSoeWrite ... 267
9.3.3 IL_ECATSoeCommand ... 269
9.4 Diagnostics ... 271
9.4.1 IL_ECATGetNumConfigSlaves ... 271
9.4.2 IL_ECATGetNumOnlineSlaves ... 272
9.4.3 IL_ECATGetConfigEthercatAddrList ... 273
9.4.4 IL_ECATGetOnlineEthercatAddrList ... 275
9.4.5 IL_ECATGetSlaveConfigInfo ... 276
9.4.6 IL_ECATGetSlaveOnlineInfo ... 278
9.4.7 IL_ECATMasterState ... 280
9.4.8 IL_ECATMasterBusStatistic ... 281
9.4.9 IL_ECATRemoteSlaveState ... 283

Page
9.4.10 IL_ECATRemoteSlaveSetErrorCounterPeriod ... 285
9.4.11 IL_ECATRemoteSlaveGetErrorCounterPeriod ... 286
9.4.12 IL_ECATRemoteSlaveGetErrorCounter ... 288
9.4.13 IL_ECATRemoteSlaveClearErrorCounter ... 289
9.4.14 IL_ECATCoeReadDiagHistory ... 290
9.5 State machine ... 295
9.5.1 IL_ECATMasterGetBusState ... 295
9.5.2 IL_ECATMasterSetBusState ... 296
9.5.3 IL_ECATRemoteSlaveGetDevState ... 298
9.5.4 IL_ECATRemoteSlaveSetDevState ... 299
9.6 Auxiliary functions (utilities) ... 301
9.6.1 IL_ECATSoeIdnCoding ... 301
9.6.2 IL_ECATSoeAttributeDecoding ... 301
9.6.3 IL_ECATMasterStateDecoding ... 302
9.6.4 IL_ECATSlaveStateDecoding ... 302
9.7 Data types (DataTypes) ... 303
9.7.1 IL_BUSMASTER ... 303
9.7.2 IL_ECAT_ADDRESS_TYPE ... 303
9.7.3 IL_ECAT_BASE_DATA_TYPE ... 303
9.7.4 IL_ECAT_COE_DIAG_HISTORY_FLAGS ... 305
9.7.5 IL_ECAT_COE_DIAG_HISTORY_STRUCT ... 305
9.7.6 IL_ECAT_COE_DIAG_MSG_EXT ... 306
9.7.7 IL_ECAT_COE_DIAG_MSG_RAW ... 307
9.7.8 IL_ECAT_COE_DIAG_TYPE ... 308
9.7.9 IL_ECAT_DL_STATUS ... 308
9.7.10 IL_ECAT_ESC_FEATURES ... 308
9.7.11 IL_ECAT_MASTER_BUS_STATISTIC ... 309
9.7.12 IL_ECAT_MASTER_STATE ... 309
9.7.13 IL_ECAT_MBX_CONFIG_PROTOCOLS ... 310
9.7.14 IL_ECAT_MBX_ONLINE_PROTOCOLS ... 311
9.7.15 IL_ECAT_MBX_SIZE_INFO ... 311
9.7.16 IL_ECAT_MEMORY_INFO ... 311
9.7.17 IL_ECAT_PORT_ERROR_COUNTER ... 311
9.7.18 IL_ECAT_PORT_STATE ... 312
9.7.19 IL_ECAT_SLAVE_CONFIG_INFO ... 313
9.7.20 IL_ECAT_SLAVE_ERROR_COUNTER ... 314
9.7.21 IL_ECAT_SLAVE_IDENTITY_INFO ... 314
9.7.22 IL_ECAT_SLAVE_ONLINE_INFO ... 314
9.7.23 IL_ECAT_SLAVE_STATE ... 315
9.7.24 IL_ECAT_SOE_ATTRIBUTE ... 316
9.7.25 IL_ECAT_SOE_DATA_LENGTH ... 316
9.7.26 IL_ECAT_SOE_DATA_STATUS ... 317
9.7.27 IL_ECAT_SOE_DATA_TYPE ... 317
9.7.28 IL_ECAT_SOE_ELEMENT ... 317
9.7.29 IL_ECAT_SOE_PARAM_TYPE ... 318
9.7.30 IL_ECAT_STATE ... 318

Page

9.8 Error codes ... 318
9.8.1 EtherCAT error table ... 318

10 RIL_MappingList.library ... 321
10.1 Introduction and overview ... 321
10.2 POUs ... 323
10.2.1 IL_SlaveMapListInit ... 323
10.2.2 IL_SlaveMapListAddEntry ... 324
10.3 Data types (DataTypes) ... 327
10.3.1 IL_ADDRESSTYPE ... 327
10.3.2 IL_FIELDBUSOBJECT ... 327
10.3.3 IL_FIELDBUSTYPE ... 328
10.3.4 IL_BUSSLAVE ... 329

11 RIL_SercosIII.library ... 331
11.1 Introduction and overview ... 331
11.2 Acyclic services (AcyclicCommunication) ... 333
11.2.1 IL_SIIISvcRead ... 333
11.2.2 IL_SIIISvcWrite ... 338
11.3 Acyclic services/parameters (AcyclicCommunication/Parameter) ... 341
11.3.1 IL_SIIICommand ... 341
11.3.2 IL_SIIIReadParameter ... 348
11.3.3 IL_SIIIReadRealParameter ... 352
11.3.4 IL_SIIIReadStringParameter ... 354
11.3.5 IL_SIIIWriteParameter ... 356
11.3.6 IL_SIIIWriteRealParameter ... 359
11.3.7 IL_SERCOS_DATA_STATUS ... 361
11.4 Backing up and restoring (BackupRestore) ... 362
11.4.1 IL_SIIIParameterBackup ... 362
11.4.2 IL_SIIIParameterRestore ... 368
11.4.3 IL_BACKUP_ERRORS ... 374
11.4.4 IL_BACKUP_OPTIONS ... 374
11.5 Diagnostics ... 374
11.5.1 IL_SIIIGetConfigSlaveAdrList ... 374
11.5.2 IL_SIIIGetDiagSlaveAdrList ... 376
11.5.3 IL_SIIIGetLiveSlaveAdrList ... 377
11.5.4 IL_SIIIMasterState ... 379
11.5.5 IL_SIIIMasterStateDetails ... 380
11.5.6 IL_SIIIRemoteSlaveState ... 381
11.5.7 IL_SIIIRemoteSlaveStateDetails ... 383
11.5.8 IL_SIII_MASTER_STATE_DETAILS ... 385
11.5.9 IL_SIII_REMOTE_SLAVE_STATE_DETAILS ... 385
11.5.10 IL_SIII_REMOTE_SLAVE_STATE_EX ... 388
11.5.11 IL_SIII_REMOTE_SLAVE_STATE_EX_BITS ... 388
11.5.12 IL_SIII_DEVICE_STATUS ... 389

Page

11.5.13 IL_SIII_DEVICE_STATUS_UNION ... 389
11.5.14 IL_SIII_DEVICE_STATUS_UNION_BITS ... 390
11.5.15 IL_SIII_TOPOLOGY_PORT_STATUS ... 390
11.5.16 IL_SIII_DIAGNOSTIC_NUMBER ... 391
11.5.17 IL_SIII_DIAGNOSTIC_NUMBER_R ... 392
11.5.18 IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP ... 392
11.5.19 IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE ... 392
11.5.20 IL_SIII_DIAGNOSTIC_NUMBER_CLASS ... 393
11.5.21 IL_SIII_DEVICE_TYPE ... 393
11.5.22 IL_SIII_DRIVE_DIAG ... 393
11.5.23 IL_SIII_DRV_ERROR ... 394
11.5.24 IL_SIII_DRV_ERROR_BITS ... 394
11.5.25 IL_SIII_DRV_WARNING ... 395
11.5.26 IL_SIII_DRV_WARNING_BITS ... 395
11.5.27 IL_SIII_DRV_MESSAGE ... 396
11.5.28 IL_SIII_DRV_MESSAGE_BITS ... 396
11.5.29 IL_SIII_IO_DIAG ... 397
11.5.30 IL_SIII_IO_DIAG_STATUS_CODE_TYPE ... 397
11.5.31 IL_SIII_IO_DIAG_STATUS_CODE ... 398
11.5.32 IL_FBUS_COMMUNICATION_STATE ... 405
11.6 Auxiliary functions (utilities) ... 405
11.6.1 IL_SIIIAttribute ... 405
11.6.2 IL_SIIIElementsToIdn ... 408
11.6.3 IL_SIIIGetSercosPhase ... 409
11.6.4 IL_SIIISetSercosPhase ... 410
11.6.5 IL_SIIIStringToIdn ... 412
11.6.6 IL_SIII_DATA_LENGTH ... 413
11.6.7 IL_SIII_DATA_TYPE ... 414
11.6.8 IL_SIII_ELEMENT ... 414
11.6.9 IL_SIII_PARAM_TYPE ... 414
11.6.10 IL_SIII_SERCOS_PHASE ... 415
11.6.11 IL_BUSMASTER ... 415

12 RMB_SercosIII_Util.library ... 417
12.1 Introduction and overview ... 417
12.2 Drives ... 422
12.2.1 MB_SIIIPlcDriveType01 ... 422
12.2.2 MB_SIIIPlcDriveVelocityType01 ... 426
12.2.3 MB_SIIIPlcDriveTorqueType01 ... 430
12.2.4 MB_SIIIPlcDrivePositioningType01 ... 434
12.2.5 MB_SIIIPlcDrivePosInterpolationType01 ... 440
12.2.6 MB_SIII_CONTROLWORD ... 444
12.2.7 MB_SIII_STATUSWORD ... 445
12.2.8 MB_SIII_DRIVE_SHUTDOWN_MODE ... 445
12.2.9 MB_SIII_HOMING_MODE ... 445
12.2.10 MB_SIII_POSCTRLWORD ... 446

Page

12.2.11 MB_SIII_IN_PROC_VELOCITY ... 447
12.2.12 MB_SIII_OUT_PROC_VELOCITY ... 447
12.2.13 MB_SIII_IN_PROC_TORQUE ... 447
12.2.14 MB_SIII_OUT_PROC_TORQUE ... 447
12.2.15 MB_SIII_IN_PROC_POSITIONING ... 448
12.2.16 MB_SIII_OUT_PROC_POSITIONING ... 448
12.2.17 MB_SIII_IN_PROC_POS_INTERPOLATION ... 448
12.2.18 MB_SIII_OUT_PROC_POS_INTERPOLATION ... 448
12.3 PowerSupply ... 449
12.3.1 MB_SIIIPowerSupplyDriveIntegrated ... 450
12.3.2 MB_SIIIPowerSupplyStandalone ... 458
12.3.3 MB_SIIIPowerSupplyDcDcIslandGrid ... 465
12.3.4 MB_POWERSUPPLY_MODE ... 472
12.3.5 MB_POWERSUPPLY_ACTUAL_MODE ... 472
12.3.6 MB_POWERSUPPLY_ERROR_MODE ... 473
12.3.7 MB_POWERSUPPLY_TARGET_MODE ... 474
12.3.8 MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID ... 475
12.3.9 MB_POWERSUPPLY_FUNCTION ... 475
12.4 RexrothEFC ... 476
12.4.1 MB_SIIIEfcCodeToIdn ... 476
12.4.2 MB_SIIIPlcDriveEfcFrequencyType01 ... 477
12.4.3 MB_EFC_FUNCTION_CODE ... 481
12.4.4 MB_SIII_IN_PROC_EFC_FREQ ... 481
12.4.5 MB_SIII_OUT_PROC_EFC_FREQ ... 481
12.4.6 MB_SIII_EFC_CONTROLWORD ... 482
12.4.7 MB_SIII_EFC_STATUSWORD ... 482
12.5 RexrothHydraulicDrives ... 483
12.5.1 MB_SIIIPlcDriveFlowControl ... 483
12.5.2 MB_SIIIPlcDrivePressureFlowControl ... 487
12.5.3 MB_SIIIPlcDrivePressureSwivelAngleControl ... 492
12.5.4 MB_SIIIPlcDrivePressureSwivelAngleSlave ... 497
12.5.5 MB_SIIIPlcDriveDirectValveControl ... 501
12.5.6 MB_SIIIPlcDriveTorqueDirectValveControl ... 505
12.5.7 MB_SIIIPlcDriveTorqueFlowControl ... 511
12.5.8 MB_SIIIPlcDrivePressureDirectValveControl ... 516
12.5.9 MB_SIIIPlcDriveVelocityAltCtrl ... 521
12.5.10 MB_SIII_HYDR_CONTROL_MODE ... 526
12.5.11 MB_SIII_IN_PROC_FLOW ... 526
12.5.12 MB_SIII_OUT_PROC_FLOW ... 527
12.5.13 MB_SIII_IN_PROC_PRESSURE_FLOW ... 527
12.5.14 MB_SIII_OUT_PROC_PRESSURE_FLOW ... 528
12.5.15 MB_SIII_IN_PROC_SWIVEL ... 529
12.5.16 MB_SIII_OUT_PROC_SWIVEL ... 529
12.5.17 MB_SIII_IN_PROC_SWIVEL_SLAVE ... 530
12.5.18 MB_SIII_OUT_PROC_SWIVEL_SLAVE ... 530
12.5.19 MB_SIII_IN_PROC_SPOOL ... 531

Page

12.5.20 MB_SIII_OUT_PROC_SPOOL ... 531
12.5.21 MB_SIII_IN_PROC_TORQUE_SPOOL ... 532
12.5.22 MB_SIII_OUT_PROC_TORQUE_SPOOL ... 532
12.5.23 MB_SIII_IN_PROC_TORQUE_FLOW ... 532
12.5.24 MB_SIII_OUT_PROC_TORQUE_FLOW ... 533
12.5.25 MB_SIII_IN_PROC_PRESSURE_SPOOL ... 533
12.5.26 MB_SIII_OUT_PROC_PRESSURE_SPOOL ... 534
12.5.27 MB_SIII_IN_PROC_VEL_ALT_CTRL ... 534
12.5.28 MB_SIII_OUT_PROC_VEL_ALT_CTRL ... 535
12.6 RexrothSytronix ... 535
12.6.1 MB_SIIIDRn5020 ... 535
12.6.2 MB_SIIIFcP5020 ... 541
12.6.3 MB_SIII_DRN_ENABLE ... 545
12.6.4 MB_SIII_DRN_VAR_VEL_ENABLE ... 545
12.6.5 MB_SIII_FCP_MASTERSLAVE ... 545
12.6.6 MB_SIII_FCP_PARAGROUP ... 546
12.6.7 MB_SIII_FCP_DUALPUMPSTATUS ... 546

13 RIL_SercosIIIDeviceReplacement.library ... 547
13.1 Introduction and overview ... 547

14 RIL_InlineIOModules.library ... 557
14.1 Introduction and overview ... 557
14.2 Analog input terminals ... 560
14.2.1 IL_AI2Sf ... 560
14.2.2 IL_AI4Ef ... 564
14.2.3 IL_AI4IU ... 569
14.2.4 IL_AI8SfIs ... 573
14.2.5 IL_Sgi2f ... 578
14.2.6 IL_Sgi2fOffset ... 584
14.2.7 IL_Temp2Uth ... 584
14.2.8 IL_Temp2Rtd ... 589
14.2.9 IL_Temp4_8Rtd ... 595
14.2.10 IL_Temp4_8RtdEf ... 601
14.3 Analog output terminals ... 607
14.3.1 IL_AO1Sf ... 607
14.3.2 IL_AO2SfBp ... 610
14.3.3 IL_AO2UI ... 614
14.4 Function terminals ... 618
14.4.1 IL_CntCount ... 618
14.4.2 IL_CntFrequency ... 624
14.4.3 IL_CntPulse ... 630
14.4.4 IL_CntTime ... 634
14.4.5 IL_Pwm2Frequency ... 639
14.4.6 IL_Pwm2Pwm ... 642

Page

14.4.7 IL_Pwm2PulseDirection ... 646
14.4.8 IL_Pwm2SinglePulse ... 649
14.4.9 IL_Ssi ... 652
14.4.10 IL_SsiIn ... 667
14.4.11 IL_IncIn ... 672
14.5 Communication modules ... 677
14.5.1 IL_RsCom ... 677
14.6 Data types ... 684
14.6.1 IL_INLN_AI_CONFIG ... 684
14.6.2 IL_INLN_AI_RANGE ... 684
14.6.3 IL_INLN_AI_DATA ... 685
14.6.4 IL_INLN_DIAG_CODE ... 685
14.6.5 IL_INLN_CNT_INCONFIG ... 686
14.6.6 IL_INLN_TEMP_CONFIG ... 686
14.6.7 IL_INLN_TEMP_CONNECTIONTYPE ... 686
14.6.8 IL_INLN_TEMP_R0 ... 687
14.6.9 IL_INLN_TEMP_SENSOR ... 687
14.6.10 IL_INLN_TEMP_EF_CONFIG ... 688
14.6.11 IL_INLN_TEMP_EF_ODS ... 688
14.6.12 IL_INLN_TEMP_EF_R0 ... 688
14.6.13 IL_INLN_TEMP_EF_SENSOR ... 689
14.6.14 IL_INLN_RS_CONFIG ... 689
14.6.15 IL_INLN_RS_PROTOCOL ... 690
14.6.16 IL_INLN_RS_BAUDRATE ... 690
14.6.17 IL_INLN_RS_DATAWIDTH ... 691
14.6.18 IL_INLN_RS_INTERFACE ... 691
14.6.19 IL_INLN_RS_MODULETYPE ... 692
14.6.20 IL_INLN_RS_DIAG ... 692

15 RIH_Safety.library ... 693

16 RIH_Safety_ProfibusDP.library ... 695
16.1 IH_PBDPAck_PSDx_Inline ... 695
16.2 IH_PBDPAck_PSDx_S20 ... 700

17 RIH_Safety_ProfinetIO.library ... 707
17.1 IH_PNIOAck_PSDx_S20 ... 707
17.2 IH_PNIOAck_PSDx_Inline ... 713

18 RIH_Safety_SercosIII.library ... 719
18.1 IH_SIIIAck_SSDx_S20 ... 719

19 RIH_Dali.library ... 725
19.1 RIH_Dali, general information ... 725

Page

19.2 IH_DaliDim ... 726
19.3 IH_DaliAddress ... 728
19.4 IH_DaliCallScene ... 730
19.5 IH_DaliRemoveFromGroup ... 732
19.6 IH_DaliDirectPower ... 734
19.7 IH_DaliRemoveFromScene ... 736
19.8 IH_DaliQueryStatus ... 738
19.9 IH_DaliReadWrite ... 742
19.10 IH_DaliQueryActualLevel ... 744
19.11 IH_DaliAddToGroup ... 747
19.12 IH_DaliAddToScene ... 749
19.13 IH_DaliOff ... 751
19.14 IH_DaliOn ... 753
19.15 IH_DaliSearchAddress ... 755
19.16 IH_DaliQuerySceneLevel ... 757
19.17 IH_DaliProgramShortAddress ... 759
19.18 IH_DaliStoreParameter ... 763
19.19 IH_DaliReset ... 766
19.20 IH_DaliQueryDTR ... 769
19.21 IH_DaliQueryParameter ... 771
19.22 IH_DaliQueryGroup ... 774
19.23 IH_DaliQueryRandomAddress ... 777
19.24 RIH_Dali library data types ... 780
19.24.1 IH_INLN_DALI_TOKEN ... 780
19.24.2 IH_INLN_DALI_ADDRTYPE ... 781
19.24.3 IH_INLN_DALI_DIMMODE ... 781
19.24.4 IH_INLN_DALI_LIGHTLEVEL ... 781
19.24.5 IH_INLN_DALI_OPMODE ... 782
19.24.6 IH_INLN_DALI_PARAMETER ... 783
19.25 Error Handling of Library RIH_Dali ... 783

20 RIH_Modbus.library ... 785
20.1 RIH_Modbus, general information ... 785
20.2 IH_MBGetCommEventCounter ... 787
20.3 IH_MBGetCommEventLog ... 790
20.4 IH_MBMaskWriteRegister ... 794
20.5 IH_MBReadCoils ... 797
20.6 IH_MBReadDeviceIdentification ... 801
20.7 IH_MBReadDiagnostics ... 807
20.8 IH_MBReadDiscreteInputs ... 810
20.9 IH_MBReadExceptionStatus ... 814
20.10 IH_MBReadFIFOQue ... 816
20.11 IH_MBReadFileRecord ... 819
20.12 IH_MBReadHoldingRegisters ... 823
20.13 IH_MBReadInputRegisters ... 827
20.14 IH_MBReadWriteMultipleRegisters ... 831

Page

20.15 IH_MBRTUMaster ... 835
20.16 IH_MBRTUSlave ... 840
20.17 IH_MBWriteFileRecord ... 846
20.18 IH_MBWriteMultipleCoils ... 851
20.19 IH_MBWriteMultipleRegisters ... 854
20.20 IH_MBWriteSingleCoil ... 858
20.21 IH_MBWriteSingleRegister ... 861
20.22 Data types ... 864
20.22.1 IH_INLN_MB_DATA ... 864
20.22.2 IH_INLN_MB_DEV_IDCODE ... 864
20.22.3 IH_INLN_MB_FIFODATA ... 865
20.22.4 IH_INLN_MB_CONFIG ... 865
20.22.5 IH_INLN_MB_TOKEN ... 866
20.22.6 IH_INLN_MB_SUBREQUESTS ... 866
20.22.7 IH_INLN_MB_RECORD ... 866
20.22.8 IH_INLN_MB_EVENTLOG ... 867
20.23 Error Handling of Library RIH_Modbus ... 867

21 RIL_IoLink.library ... 871
21.1 Introduction and overview ... 871
21.2 General information on the IO-Link ... 871
21.3 IL_SIII_IoLinkProxy ... 873
21.4 IL_IoLinkCall ... 875
21.5 IL_IoLinkErrorText ... 880
21.6 IL_IoLinkIdent ... 880
21.7 RexrothHydraulicValve ... 882
21.7.1 IL_IoLinkDirectValveControlType01 ... 882
21.7.2 IL_IOL_OUT_PROC_SPOOL_TYPE01 ... 885
21.7.3 IL_IOL_IN_PROC_SPOOL_TYPE01 ... 886

22 Modbus RTU - Libraries ... 889
22.1 IoDrvModbusComPort ... 889
22.2 ModbusSlaveComPort ... 889
22.3 IoDrvModbusTCP ... 890
22.4 ModbusTCPSlave ... 890
22.5 IoDrvModbusSerialSlave ... 891
22.6 IoDrvModbusTCPSlave ... 892
22.7 ModbusServer ... 892
22.8 ModbusRequest ... 893

23 RIH_IndraControl_S20.library ... 897
23.1 Introduction and overview ... 897
23.2 Error codes ... 900
23.3 Diagnostics ... 900
23.3.1 IH_S20IOModuleStateDetails ... 900

Page

23.3.2 IH_S20IOState ... 901
23.3.3 IH_S20IOStateDetails ... 902
23.4 I&M (Identification and Maintenance) (IM Data) ... 903
23.4.1 IH_S20ModuleIdentDetails ... 903
23.5 Acyclic communication (Acyclic Communication) ... 904
23.5.1 IH_S20ComConfiguration ... 904
23.5.2 IH_S20ComInterfaceType02 ... 905
23.5.3 IH_S20ComInterface ... 906
23.6 Analog input modules (AnalogIO) ... 912
23.6.1 IH_S20AI4IU ... 912
23.6.2 IH_S20AI8 ... 916
23.6.3 IH_S20AI4RTD ... 918
23.6.4 IH_S20AI4UTH ... 920
23.6.5 IH_S20AI8RTD ... 922
23.6.6 IH_S20AI8UTH ... 924
23.7 Analog output modules (AnalogIO) ... 926
23.7.1 IH_S20AO8 ... 926
23.7.2 IH_S20AO4 ... 928
23.8 Technology modules (TechnologyIO) ... 931
23.8.1 IH_S20SsiAo1 ... 931
23.8.2 IH_S20CntInc ... 935
23.8.3 IH_S20RSCom ... 940

24 RIH_IndraControl_S20_KNX ... 949
24.1 General information ... 949
24.2 Application example ... 956

25 BACnet libraries ... 967
25.1 Overview ... 967
25.2 Programming examples ... 968
25.2.1 Server: NotificationClass → SetRecipientList ... 968
25.2.2 Server/client: IBACnetEventConsumer → BACnetEventCallback ... 968

26 Service and support ... 971

Index ... 973

1 About this documentation

1.1 Change record

Editions of this documentation

Edition	Release date	Note
01	2019-03	First edition
02	2019-06	Neu: RIH_IndraControl_S20_KNX, RIL_SercosIIIDeviceReplacement Update: MB_SIIIPowerSupplyType03, MB_SIIIPowerSupplyType04
03	2019-10	New: chapter 12.3.3 "MB_SIIIPowerSupplyDcDcIslandGrid" on page 465 Update: MB_SIIIPowerSupplyType03 changed in MB_SIIIPowerSupplyDriveIntegrated, MB_SIIIPowerSupplyType04 changed in MB_SIIIPowerSupplyStandalone, chapter 24 "RIH_IndraControl_S20_KNX" on page 949, MB_SIIIPowerSupplyType01 and MB_SIIIPowerSupplyType02 removed BACnet: Note added
04	2020-06	Update: chapter 24 "RIH_IndraControl_S20_KNX" on page 949
05	2026-01	Revision: • chapter 12.3.1 "MB_SIIIPowerSupplyDriveIntegrated" on page 450 (Defdb00212868) • chapter 12.3.2 "MB_SIIIPowerSupplyStandalone" on page 458 (Defdb00212868)

Tab. 1-1: Revision history

1.2 Validity of the documentation

Overview on target groups and product phases

In the following illustration, the framed activities, product phases and target groups refer to the present documentation.

Example: In the product phase "Commissioning", the target group "Technologist" can execute the activities "Optimize" and "Test" using this documentation.

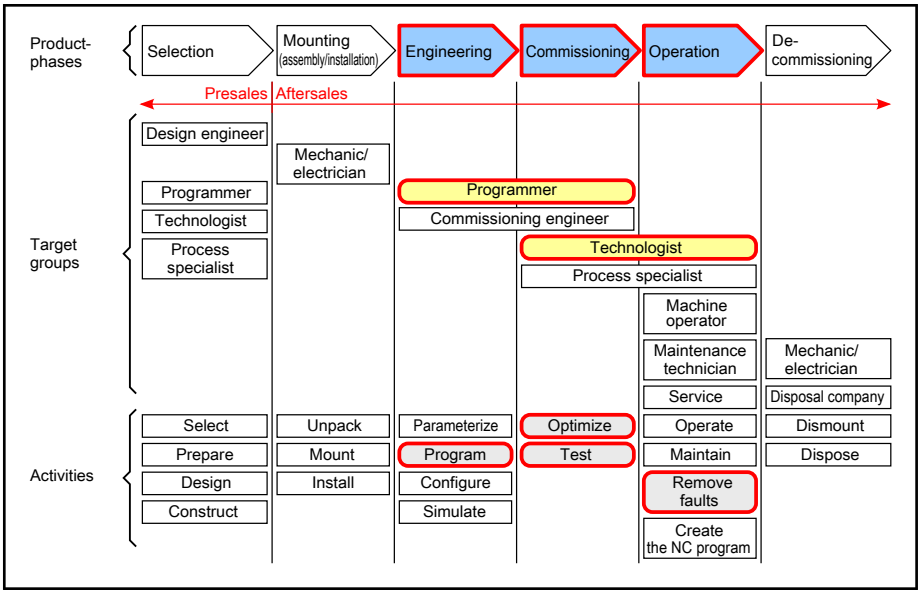


Fig. 1-1: Assigning the present documentation to the target groups, product phases and activities of the target group

Target group This documentation is intended to inform on the available field bus libraries of MLC or MTX controls.

Application phases This documentation supports the user during the following phases:

- Field bus application in the PLC user program

1.3 Libraries and target systems

The **IndraLogic 2G Field Bus Libraries** are described with regard to the systems **MLC** and **MTX**:

Name	Description	MLC	MTX
RIL_ProfibusDP_02, page 31	Profibus DP master V1 services, diagnostics, sync and freeze	(x)	(x)
RIL_ProfibusDPSlave, page 57	Profibus DP slave V1 services, diagnostics	x	x
RIL_ProfinetIO, page 69	Diagnostic and communication services for Profinet I/O controller	x	x
RIL_ProfinetIODevice, page 205	Diagnostic and communication services for Profinet I/O device	x	x
RIL_EtherNetIP, page 217	Diagnostic and communication services for Ethernet IP scanner	x	x
RIL_EtherNetIPAdapter, page 245	Diagnostic and communication services for Ethernet IP adapter and Ethernet IP adapter (Engineering interface)	x	x
RIL_EtherCAT, page 257	Interface between the PLC programming environment and the EtherCAT master	x	
RIL_MappingList, page 321	Mapping table for acyclic access on field bus slaves (DP slave, Profinet I/O device and ENIP adapter)	x	x
RIL_SercosIII, page 331	Diagnostic and communication services for Sercos III	x	x
RMB_SercosIII_Util, page 417	Simplified axis control via the Sercos bus. The drive is created as PLC device	x	x
RIL_SercosIIIDeviceReplacement, page 547	Allows a device exchange without PC	x	x
RIL_InlineIOModules, page 557	Access options of an IEC program to Rexroth Inline I/O modules	x	x

Name	Description	MLC	MTX
RIH_Safety, page 693	Supports the access to the standard control to data of the SafeLogic safety control by providing data types	x	x
RIH_Safety_ProfibusDP, page 695	Contains an acknowledgement function block for Safety I/O modules of Bosch Rexroth operated at SafeLogic (inline)	x	x
RIH_Safety_ProfinetIO, page 707	Contains acknowledgement function blocks for Safety I/O modules of Bosch Rexroth operated at SafeLogic (inline and S20)	x	x
RIH_Safety_SercosIII, page 719	Contains an acknowledgement function block for Safety I/O modules of Bosch Rexroth operated at SafeLogic (S20)	x	x
RIH_Dali, page 725	To be used with the Inline modules **R-IB IL DALI/PWR-PAC** (R911172212) and **R-IB IL DALI-PAC** (R911172213). The Inline module can be connected to the control or to an Inline bus coupler (Profibus, Profinet, Sercos)	x	x
RIH_Modbus, page 785	Provides function blocks to communicate with the Modbus RTU protocol as master or client. To be used with the Inline modules **R-IB IL RS232 PRO**, **R-IB IL RS485/422 PRO** or **R-IB IL RS-Uni** (Inline modules for a serial data transfer). The Inline modules can be connected to the control or an Inline bus coupler (Profibus, Profinet, Sercos)	x	x
RIL_IoLink.library, page 871	Includes field bus-independent function blocks to control IO-Link devices	x	x
RIH_IndraControl_S20, page 897	Access options of the IEC program to the IndraControl S20 system using function blocks	x	x
RIH_IndraControl_S20_KNX, page 949	Access options of the PLC program to the IndraControl S20 KNX module using function blocks	x	x
BACnet.library, page 967	Interface between the PLC programming environment and the BACnet stack	x	

x Library exists/is available for the system
(x) Library is being processed

Tab. 1-2: IndraLogic 2G field bus libraries

1.4 Using safety instructions

1.4.1 Structure of the safety instructions

The safety instructions are structured as follows:

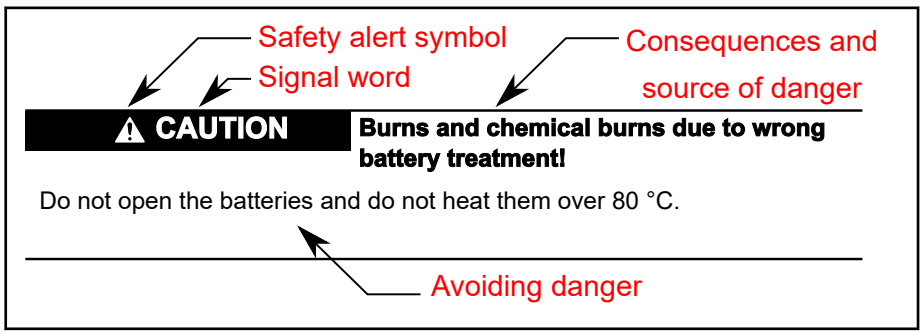


Fig. 1-2: Structure of the safety instructions

1.4.2 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.

1.4.3 Symbols used

Pointers are displayed as follows:

 This is a note.

Tips are displayed as follows:

 This is a tip.

1.4.4 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.

1.5 Required and supplementing documentation

The documents listed below contain additional information on this subject.

IndraWorks 14VRS IndraLogic 2G PLC Programming System DOK-IWORKS-IL2GPRO*V14-APRS-EN-P, R911343571 This documentation describes the PLC programming tool IndraLogic 2G and its use. It includes the basic use, first steps, visualization, menu items and editors.
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 IndraMotion MLC and IndraMotion MTX systems. The focus of this documentation is on the configuration, parameterization, commissioning and the diagnostics of the different periphery connections. It is the basis for the online help.
IndraWorks 14VRS Basic Libraries IndraLogic 2G DOK-IL*2G*-BASLIB**V14-LIRS-EN-P, R911343920 This documentation describes the system-comprehensive PLC libraries.
IndraMotion 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_CommonTypes, ML_Base and ML_PLCopen libraries for the IndraMotion MLC. The documentation also includes the error reactions of the function blocks.
Automation Terminals of the Rexroth Inline Product Range DOK-CONTRL-ILSYSINS***-AWRS-EN-P, R911317021 This documentation describes the automation terminals of the Rexroth Inline product family.
Rexroth Inline Bus Coupler for Sercos with Digital Inputs and Outputs R-IB IL S3 BK DI8 DO4-PAC DOK-CONTRL-ILS3BKDI8DO-KBRS-EN-P , R911170560 This documentation describes a bus coupler provided for a Sercos III network. It is the connection to the Inline installation system.

Rexroth Inline bus coupler for Profibus DP with digital inputs and outputs R-IL PB BK DI8 DO4-PAC DOK-CONTRL-ILPBBKDI8DO-AWRS-EN-P, R911170476 This documentation describes a bus coupler provided for a Profibus DP network. It is the connection to the Inline installation system.
Rexroth Inline bus coupler for Profinet I/O with digital inputs and outputs R-IL PN BK DI8 DO4-PAC DOK-CONTRL-ILPNBKDIO**-DARS-EN-P, R911328682 This documentation describes a bus coupler provided for a Profinet I/O network. It is the connection to the Inline installation system.
Rexroth Inline Bus Coupler for Ethernet IP with Digital Inputs and Outputs R-IL ETH BK DI8 DO4 2TX-PAC DOK-CONTRL-ILETHBKDIO*-DARS-EN-P, R911328680 This documentation describes a bus coupler provided for a Ethernet IP network. It is the connection to the Inline installation system.

1.6 Required and supplementing MLC documentations

Documentation titles with type codes and part numbers

System overview
MLC 15VRS System Overview DOK-MLC***-SYSTEM**V15-PRRS-EN-P, R911394936 This documentation provides an overview of the hardware/software components of the automation system MLC in the mentioned version. It helps assembling a system.

Tab. 1-3: MLC documentation overview - System overview

First steps
MLC 15VRS First Steps DOK-MLC***-F*STEP**V15-CORS-EN-P, R911398631 This documentation describes the first steps of the MLC and the RobotControl. It includes the hardware and software prerequisites as well as the creation of a project.
MLC 15VRS Project Conversion DOK-MLC***-PROCONV*V15-APRS-EN-P, R911396595 This documentation describes the project conversion of the MLC system.

Tab. 1-4: MLC documentation overview - First steps

Engineering
IndraWorks 15VRS Software Installation
DOK-IWORKS-SOFTINS*V15-CORS-EN-P, R911393450
This documentation describes the IndraWorks installation.
IndraWorks 15VRS Engineering
DOK-IWORKS-ENGINEE*V15-APRS-EN-P, R911393303
This documentation describes the use of IndraWorks in which the Rexroth Engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.
MLC/ILC 15VRS Functional Description
DOK-MLC***-FUNC****V15-APRS-EN-P, R911393305
This documentation describes wizards, context menus, dialogs, control commissioning, device configuration and functionalities of the MLC/ILC system.

Tab. 1-5: MLC documentation overview - Engineering

Diagnostics and service
MLC/ILC 15VRS Diagnostics
DOK-MLC***-DIAG****V15-RERS-EN-P, R911393324
This documentation includes all control diagnostics implemented into the control system MLC/ILC.
MLC/ILC 15VRS Parameters
DOK-MLC***-PARAM***V15-RERS-EN-P, R911393322
This documentation describes the MLC/ILC parameters as well as the interaction between parameterization and programming. It includes the axis parameters, control parameters, kinematic parameters, touch probe parameters and programmable limit switch parameters.
WebAssistant
DOK-GENERL-WEB*ASSIST*-APRS-EN-P, R911381469
This documentation describes the WebAssistant. The WebAssistant is a web-based diagnostic tool used to access a control system via an Ethernet high-speed connection. The WebAssistant allows OEMs, end users and service engineers to access and to remotely diagnose a system. A PC with Internet Explorer version 11 or higher, Firefox version 45 or above or Google Chrome version 54 or above is required. The following control variants are supported: • MLC XM21/XM22/XM42/VPx

Tab. 1-6: MLC documentation overview - Diagnostics and service

Firmware
Rexroth IndraDrive MPx-21, Functions [1)]
DOK-INDRV*-MP*-21VRS**-APRS-EN-P, R911385758
This documentation describes all functional properties of the IndraDrive firmware in the variants MPB-20, MPM-20, MPC-20 and MPE-20.
IndraDrive MPx-16 to MPx-21 and PSB, Parameters
DOK-INDRV*-GEN1-PARA**-RERS-DE-P, R911328651
This documentation describes all parameters implemented in the firmware for drive controllers of the IndraDrive family. It supports the parameterization of the drive controllers.

Rexroth IndraDrive MPx-16 to MPx-21 und PSB, Diagnostics
DOK-INDRV*-GEN1-DIAG**-RERS-DE-P, R911326738
This documentation describes all diagnostics implemented in the following firmwares: • Drive Controller Firmwares MPx-16 to MPx-21 • PSB firmware variants for "HMU05" and "KMV03" type supply units • Firmware variants for "HMV01" type supply units. It supports the operating crew as well as the programmer during troubleshooting.
Hardware
Rexroth IndraDrive Control Units CSB02, CSE02, CSH02, CDB02
DOK-INDRV*-Cxx02******-PRRS-EN-P, R911338962
Rexroth IndraDrive Cs Drive Systems With HCS01
DOK-INDRV*-HCS01******-PRRS-EN-P, R911322210
Rexroth IndraDrive Mi, Drive Systems with KCU02 KSM02, KMS02/03, KMV03
DOK-INDRV*-KCU02+KSM02-PRxx-EN-P, R911335703
Rexroth IndraDrive ML Drive Systems with HMU05
DOK-INDRV*-Hxx05******-PRxx-EN-P, R911344279

1) Being prepared

Tab. 1-7: Overview on documentations for drive controllers

Firmware
ctrlX DRIVE Runtime AXS-V-02 Functions
DOK-XDRV**-AXS-02VRS**-APRS-EN-P, R911398021
This documentation describes all functional properties of the ctrlX DRIVE firmware AXSV02. It applies to the single-axis converter, the multi-axis inverters and the regenerative supply unit.
ctrlX DRIVE, Diagnostic Messages of Runtime AXS
DOK-XDRV**-GEN2-DIAG**-RERS-EN-P, R911383776
This documentation describes all diagnostic options of the device states of the ctrlX DRIVE product line. Furthermore, the documentation contains the descriptions of the diagnostic messages implemented in the FWA-XD1-AXS-V-02 firmware/ runtime to support machine operators and installation programmers with troubleshooting.
ctrlX DRIVE, Parameters of Runtime AXS
DOK-XDRV**-GEN2-PARA**-RERS-EN-P, R911383778
This documentation describes all parameters implemented into the firmware for drive controllers of the ctrlX DRIVE product range. It supports the parameterization of the drive controllers.
Hardware
ctrlX DRIVE, Drive Systems
DOK-XDRV**-X**********-PRRS-EN-P, R911386579
DOK-XDRV**-X******ENH1-PRRS-EN-P; R911390471
• Overview on the drive system • Description of the allowed combinations of system components • Selection of the system components of the drive system • Specification applying to all components (ambient and operating conditions) • Application description of system characteristics

Tab. 1-8: Overview on documentations for the ctrlX DRIVE drive controllers

PLC
IndraWorks 15VRS PLC Programming System IndraLogic 2G
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.
IndraWorks 15VRS Basic Libraries IndraLogic 2G
DOK-IL*2G*-BASLIB**V15-LIRS-EN-P, R911398633
This documentation describes the system-comprehensive PLC libraries.
MLC 15VRS PLCopen Libraries
DOK-MLC***-FUNLIB**V15-LIRS-DE-P, R911393307
This documentation describes the function blocks, functions and data types of the RIL_CommonTypes, ML_Base and ML_PLCopen libraries for the MLC system. The documentation also includes the error reactions of the function blocks.

Tab. 1-9: MLC documentation overview - PLC

Field buses
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.
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_ProfibusDPSlave, RIL_ProfinetIO, RIL_ProfinetIO-Device, RIL_EtherNetIPAdapter, RIL_MappingList, RIL_SERCOSIII, RIH_IndraControl_S20 including their diagnostics and error reactions of the function blocks.
Sercos System Manual for I/O Devices
DOK-CONTRL-ILS3*******-APxx-EN-P, R911333512
This documentation describes the configuration, parameterization, commissioning and diagnostics of I/O devices with a Sercos interface.

xx Corresponding edition

Tab. 1-10: MLC documentation overview - Field buses

HMI
IndraWorks 14VRS HMI
DOK-IWORKS-HMI*****V14-APRS-EN-P, R911343569
This documentation describes the functions, configuration and operation of the user interfaces IndraWorks HMI Engineering and IndraWorks HMI Operation.
IndraWorks 14VRS WinStudio 7.4
DOK-IWORKS-WINSTUD*V14-APRS-EN-P, R911341585
This "User Manual and Technical Reference Book" facilitates working with the "Rexroth WinStudio" software for optimal results. This document provides technical information and step-by-step instructions to create web-enabled HMI/SCADA programs.
MLC 15VRS HMI Connection
DOK-MLC***-HMI*****V15-APRS-EN-P, R911399270
This documentation describes the visualization systems supported by the MLC and their connection.

Tab. 1-11: MLC documentation overview - HMI

Technology
MLC 15VRS Generic Application Template DOK-MLC**-TF*GAT**V15-APRS-EN-P, R911400166 This documentation provides a structured template to the IndraLogic PLC programmer. This template can be used to add and edit the PLC programming code. It includes the template, the template wizard and use cases.
MLC 15VRS Technology Libraries DOK-MLC***-TF*LIB**V15-LIRS-EN-P, R911396136 This documentation describes the function blocks, functions and data types of the "ML_TechInterface.library", "ML_TechMotion.library", "RMB_TechCam.library" and "ML_TechBase.library". It also includes libraries for the winder functionality, register controller functionality and CrossCutter functionality.
IndraWorks 14VRS Energy Efficiency Management DOK-IWORKS-4EE*****V14-APRS-EN-P, R911339229 This documentation describes the HMI connection of the energy management system and the function blocks, functions and data types of the "RIL_4EE " library. The documentation also includes the error reactions of the function blocks.
Rexroth IndraMotion MLC 14VRS RegisterControl (Library) DOK-MLC***-REGI*CO*V14-LIRS-EN-P, R911341509 This documentation describes the inputs and outputs of the individual function blocks and provides notes on their use.
IndraMotion MLC 14VRS RegisterControl (Application Description) DOK-MLC***-REGI*CO*V14-APRS-EN-P, R911341507 This documentation describes the application of the integrated register control for a rotogravure printing machine. The components of the mark stream sensor, the HMI application and the error recovery options are described. This instruction provides information on how to operate the register control, react to errors and query diagnostics. This documentation is written for machine setters and machine operators.
Rexroth IndraMotion MLC 14VRS Winder Function Application DOK-MLC***-TF*WIND*V14-APRS-EN-P, R911341513 This application-related system documentation describes the application of the winder technology functions.
Rexroth IndraWorks 14VRS CamBuilder DOK-IWORKS-CAMBUIL*V14-APRS-EN-P, R911387570 This documentation describes the basic principles and operation of the CamBuilder, the cam editing tool.
Rexroth IndraMotion MLC 15VRS Robot Control V2 DOK-MLC***-ROCO****V15-RERS-EN-P, R911396135 This documentation provides information on the Robot Control V2. The focus is on PLCopen programming. The program structure, variables, functions, motion statements and the required system parameters are described. This documentation also includes the description of the "ML_Robot" library. Robot Control V2 is supported by a kinematic interface. The components of this interface are described in the "ML_KinTech" library.

Tab. 1-12: MLC documentation overview - Technology

Open Core Engineering - OCE
IndraLogic XLC IndraMotion MLC 14VRS First Steps MLPI (online help) DOK-XLCMLC-MLPI****V14-CORS-EN-E, R911341033 This document describes the installation and the commissioning of the MLPI interface.
IndraLogic XLC IndraMotion MLC 14VRS Automation Interface (online help) DOK-XLCMLC-AUT*INT*V14-APRS-EN-E, R911341499 This documentation describes the script-based access to IndraWorks project data via the interface of the Automation Interface.

Tab. 1-13: MLC documentation overview - Open Core Engineering - OCE

SafeLogic
IndraWorks SafeLogic 14VRS First Steps
DOK-IWORKS-SL*STEP*V14-CORS-EN-P, R911341520
This documentation describes the initial commissioning of the Safety function module and its external Safety periphery. Taking the example of a project, all required steps to commission a simple Safety application are executed.
IndraLogic XLC IndraMotion MLC 14VRS SafeLogic System Overview
DOK-XLCMLC-SL**SYS*V14-PRRS-EN-P, R911341696
This documentation describes the project planning, configuration, creation and commissioning of safety-related devices.
IndraWorks 14VRS SafeLogic - Project Configuration, Release 1.2.0.0.
DOK-IWORKS-SL**PRJ*V14-APRS-EN-P, R911341694
This documentation describes the configuration of Safety applications.
IndraControl CFL01.1-F1 Safety Function Module
DOK-CONTRL-SL**FM*****-ITRS-DE-P, R911336576
This documentation is provided with the hardware and describes the commissioning, the installation, the decommissioning and disposal of the hardware.

Tab. 1-14: MLC documentation overview - SafeLogic

Hardware
IndraControl XM21, XM22 Controls
DOK-CONTRL-IC*XM2*****-ITRS-EN-P, R911340667
This documentation describes the IndraControl XM21/XM22 controls.
IndraControl XM42 Controls
DOK-CONTRL-XM4X*CTRL**-ITRS-EN-P, R911345566
This documentation describes the IndraControl XM42 controls.
IndraControl XFE 01.1 Extension Modules Profibus, RT-Ethernet, Sercos, CAN
DOK-CONTRL-XFE**EXTMOD-ITRS-EN-P, R911345570
The documentation describes all extension modules of the XM controls.

xx Corresponding edition

Tab. 1-15: MLC documentation overview - Hardware

Hydraulics
IndraMotion MLC 14VRS Hydraulic Functions
DOK-MLC***-TF*HMOT*V14-LIRS-EN-P, R911341906
This documentation describes the function blocks, functions and data types of the libraries MH_TechHydrBase and MH_TechHydrMotion. The documentation also includes the error reactions of the function blocks.
Rexroth IndraLogic XLC IndraMotion MLC 14VRS Sequential Programming
DOK-XLCMLC-SEQPROG*V14-LIRS-EN-P, R911341909
This documentation describes the options of the sequential programming, also together with GAT and the library MH_HydrControlProg.

Tab. 1-16: MLC documentation overview - Hydraulics

1.7 Required and supplementing documentation MTX

1.7.1 Selection/compilation

Documentation titles with type codes and part numbers

MTX 15VRS System Description
DOK-MTX***-SYS*DES*V15-PRRS-EN-P, R911397742
This documentation describes the MTX control. It includes the designs, technical data, interfaces as well as the configuration of the control components.
MTX 15VRS SafeLogic System Overview
DOK-MTX***-SL**SYS*V15-PRRS-EN-P, R911398637
This documentation describes the use of the safety control SafeLogic in the MTX.
IndraControl XM42 Controls
DOK-CONTRL-XM4X*CTRL**-ITRS-EN-P, R911345566
This documentation describes the IndraControl XM42 controls.

RS Corresponding edition

Tab. 1-17: MTX documentation overview - Selecting/composing

1.7.2 Configuration

Documentation titles with type codes and part numbers

MTX 15VRS Machine Parameters
DOK-MTX***-MA*PAR**V15-RERS-EN-P, R911400176
This documentation describes structure and the adjustment of available parameters of the MTX. It also includes the functions of the NC configurator and its operation.
MTX 15VRS PLC Interface
DOK-MTX***-PLC*INT*V15-PRRS-EN-P, R911400172
This documentation describes the interface signals and the program function blocks for the integrated PLC.
MTX 15VRS Functional Description - Basics
DOK-MTX***-NC*F*BA*V15-RERS-EN-P, R911401401
This documentation describes the basic functions of the MTX. The basic commissioning steps and the control functions are provided as description and handling instruction.
MTX 15VRS Functional Description - Extension
DOK-MTX***-NC*F*EX*V15-RERS-EN-P, R911393316
This documentation describes the extended functions of the MTX. The basic commissioning steps and the control functions are provided as description and handling instruction.
MTX 15VRS Functional Description - Special Functions
DOK-MTX***-NC*F*SP*V15-RERS-EN-P, R911393309
This documentation describes the special functions of the MTX. The basic commissioning steps and the control functions are provided as description and handling instruction.

MTX Free Form Surface Milling
DOK-MTX***-FREEFORM***-APRS-EN-P, R911341435
This documentation describes the free form surface milling process wit the MTX control. CNC programs generated by a CAD/CAM system are used as basis for the entire process. Overview over the following topics: Description of the MTX or CNC function for freeform surface milling, strategy during the NC parameterization process, boundary conditions when generating CNC programs with CAM software.
MTX Conversion of MTX Projects
DOK-MTX***-PROCONV****-PRRS-EN-P, R911342484
This documentation provides support during the conversion of MTX 1.x projects to MTX 2G. This project planning manual provides information on how to convert a project and identifies potential difficulties during the conversion.

RS Corresponding edition

Tab. 1-18: MTX documentation overview - Configuring

1.7.3 Commissioning

Documentation titles with type codes and part numbers

MTX 15VRS Commissioning
DOK-MTX***-STARTUP*V15-CORS-EN-P, R911393281
This documentation describes the commissioning of the MTX control. Apart from a complete overview, commissioning and configuration of the axes and the user interface as well as the PLC data are described.
IndraWorks 15VRS Basic Libraries, IndraLogic 2G
DOK-IL*2G*-BASLIB**V15-LIRS-EN-P, R911398633
This documentation describes the system-comprehensive PLC libraries.
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.
IndraWorks 14VRS WinStudio 7.4
DOK-IWORKS-WINSTUD*V14-APRS-EN-P, R911341585
This “User Manual and Technical Reference Book” supports the user in achieving the best results with the “Rexroth WinStudio”™ software. The document provides technical information and step-by-step instructions to create web-enabled HMI/SCADA programs.
IndraWorks 15VRS Software Installation
DOK-IWORKS-SOFTINS*V15-CORS-EN-P, R911393450
This documentation describes the IndraWorks installation.
IndraWorks 15VRS Engineering
DOK-IWORKS-ENGINEE*V15-APRS-EN-P, R911393303
This documentation describes the use of IndraWorks in which the Rexroth Engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.
IndraWorks 15VRS PLC Programming System IndraLogic 2G
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.

IndraWorks 15VRS HMI
DOK-MLC***-HMI*****V15-APRS-EN-P, R911399270
This documentation describes the functions, configuration and operation of the user interfaces IndraWorks HMI Engineering and IndraWorks HMI Operation.
MTX 15VRS Shape Cutting Technology
DOK-MTX***-TECHCUT*V15-CORS-EN-P, R911399147
This documentation describes the shape cutting technology used by the machines to cut materials using an NC-controlled tool head.
MTX-integrated CAD/CAM System for Shape Cutting
DOK-MTX***-CAD*TECHCUT-CO01-EN-P, R911393320
This documentation describes the mode of operation and the area of application of the CAD/CAM nesting software Lantek Expert Inside.

RS Corresponding edition

Tab. 1-19: MTX documentation overview - Commissioning

1.7.4 Operation

Documentation titles with type codes and part numbers

MTX 12VRS Block Pre-Run
DOK-MTX***-BLK*RUN*V12-APRS-EN-P, R911334379
This documentation explains to the machine manufacturer how to setup the "Block pre-run" function at the machine for the end user.
MTX 15VRS Programming Manual
DOK-MTX***-NC**PRO*V15-RERS-EN-P, R911393318
This documentation describes the standard programming of the MTX control. Apart from the basics of NC programming, the use of NC functions according to DIN 66025 as well as the NC functions with high-level language syntax and CPL functions are described.
MTX 15VRS Standard NC Operation
DOK-MTX***-NC*OP***V15-APRS-EN-P, R911393314
This documentation describes the operation of the standard user interface of the NC control of the MTX. It includes the operation of the interface, the NC program development as well as the tool management.
MTX 15VRS Multitouch
DOK-MTX***-MULTI***V15-APRS-EN-P, R911393311
The Multitouch user interface of the NC control MTX is described in this documentation.
MTX 15VRS Standard NC Cycles
DOK-MTX***-NC*CYC**V15-PRRS-EN-P, R911394940
This documentation describes the application of the standard cycles of the different technologies for the MTX control.
MTX 15VRS NC Simulation
DOK-MTX***-NC*SIM**V15-APRS-EN-P, R911393273
This documentation describes the NC simulation for the MTX control.
MTX 15VRS Measuring Functions
DOK-MTX***-MES*FUN*V15-APRS-EN-P, R911394938
This documentation describes the measuring cycles of the MTX control.

RS Corresponding edition

Tab. 1-20: MTX documentation overview - Operating

1.7.5 OEM Engineering

Documentation titles with type codes and part numbers

MTX 13VRS Automation Interface
DOK-MTX***-AUT*INT*V13-APRS-EN-P, R911337274
This documentation describes the script-based access to IndraWorks project data via the interface of the Automation Interface. Different objects including code examples are described. The Automation Builder is also described in this manual.
MTX 15VRS OPC Communication
DOK-MTX***-OPC*COM*V15-PRRS-EN-P, R911399272
This documentation describes the syntax and the structure of the items for the communication with Bosch Rexroth devices.
IndraWorks OPC UA Communication
DOK-IWORKS-OPC*UA*****-APRS-EN-P, R911379309
This documentation describes the OPC UA communication of the MLC and MTX control systems.

RS Corresponding edition

Tab. 1-21: MTX documentation overview - OEM engineering

1.7.6 AddOns

Documentation titles with type codes and part numbers

MTX 11VRS Action Recorder
DOK-MTX***-ACR*****V11-APxx-EN-P, R911329943
This documentation describes the MTX action recorder. It includes the installation and commissioning as well as interface signals, application and operation.
MTX 15VRS Efficiency Workbench MTX cta, MTX ega
DOK-MTX***-EWB*****V15-APRS-EN-P, R911400178
This documentation describes the mode of operation and the area of application of the analysis tool MTX cta and MTX ega.
MTX Remote Condition Monitoring
DOK-MTX***-RCM*****V01-APRS-EN-P, R911334383
This documentation describes the operation of the Remote Condition Monitoring System.
MTX visIREC User Documentation
DOK-MTX***-VISIREC*V01-APRS-EN-P, R911344242
This documentation describes the analysis tool visIREC used to optimize the free form surface milling process. 2D or 4D display of path-related data. 2D or 4D display of coordinate-related data. Analysis of critical areas (path and orientation deviation). Comparing the programmed NC blocks to the interpolated NC blocks.

xx / RS Corresponding edition

Tab. 1-22: MTX documentation overview - AddOns

1.8 Names and abbreviations

Term	Explanation
EFC	Rexroth frequency converter
ENIP	Ethernet IP
PNIO	Profinet I/O
DALI	Digital Addressable Lighting Interface is a protocol for lighting and building automation according to the IEC 62386

Tab. 1-23: Terms and abbreviations

1.9 Customer feedback

Customer requests, comments or suggestions for improvement are of great importance to us. Please email your feedback on the documentations to Feedback.Documentation@boschrexroth.de. Directly insert comments in the electronic PDF document and send the PDF file to Bosch Rexroth.

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).

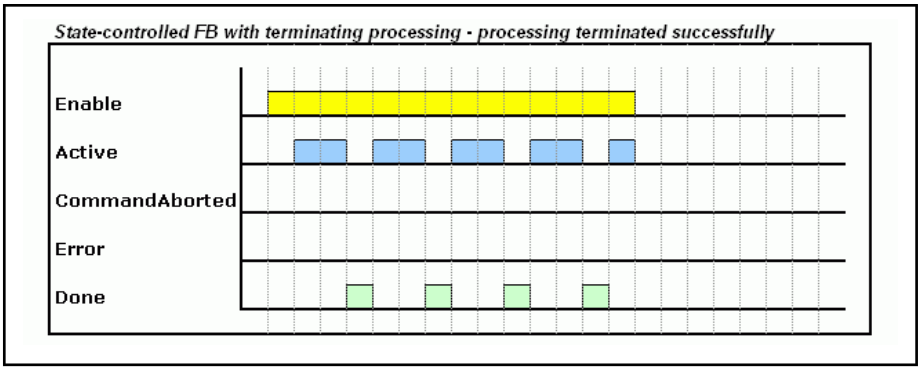


Fig. 2-1: I/O of state-controlled function blocks, final processing without "Shutdown". Processing successfully completed

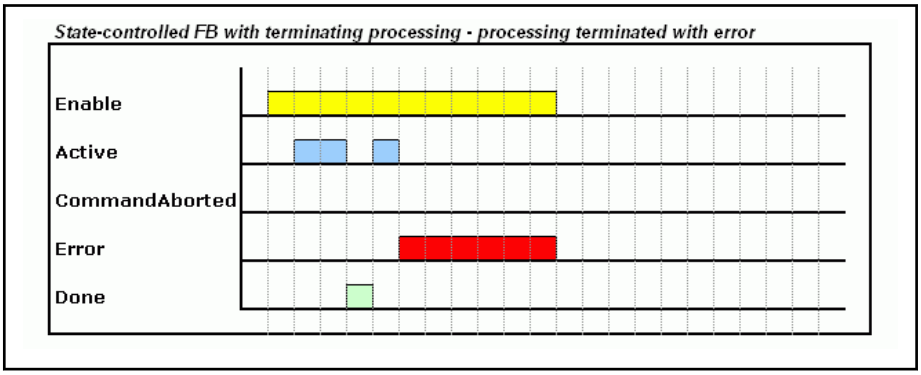


Fig. 2-2: I/O of state-controlled function blocks, final processing without "Shutdown". Processing completed with error

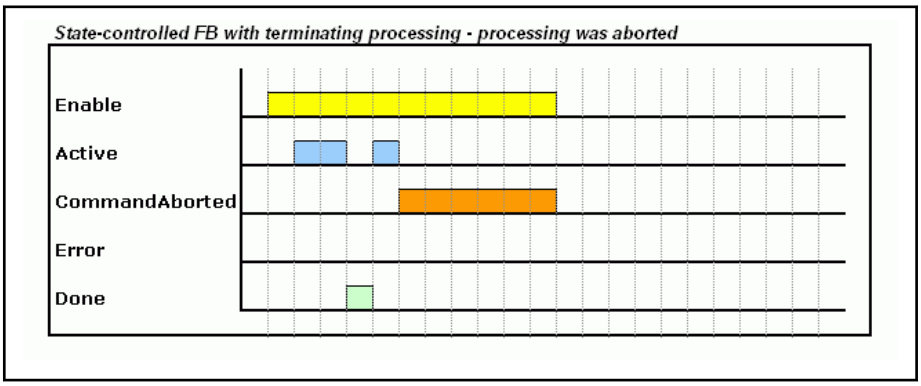


Fig. 2-3: I/O of state-controlled function blocks, final processing without "Shutdown". Processing interrupted

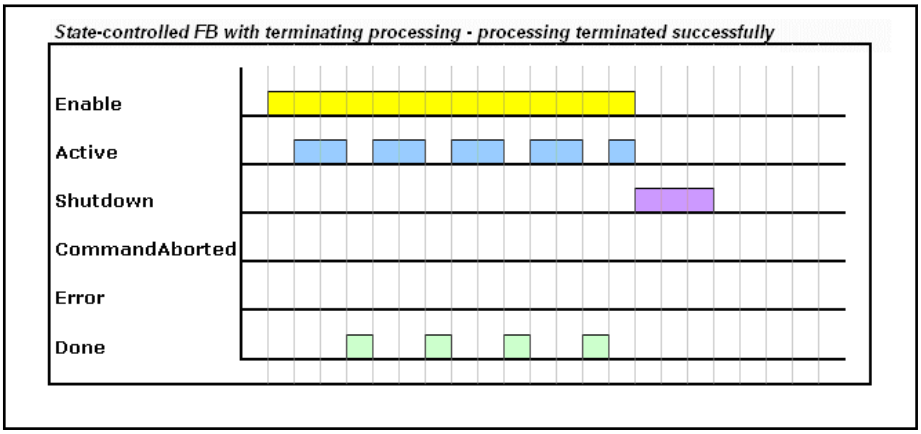


Fig. 2-4: I/O of state-controlled function blocks, final processing with "Shutdown". Processing successfully completed

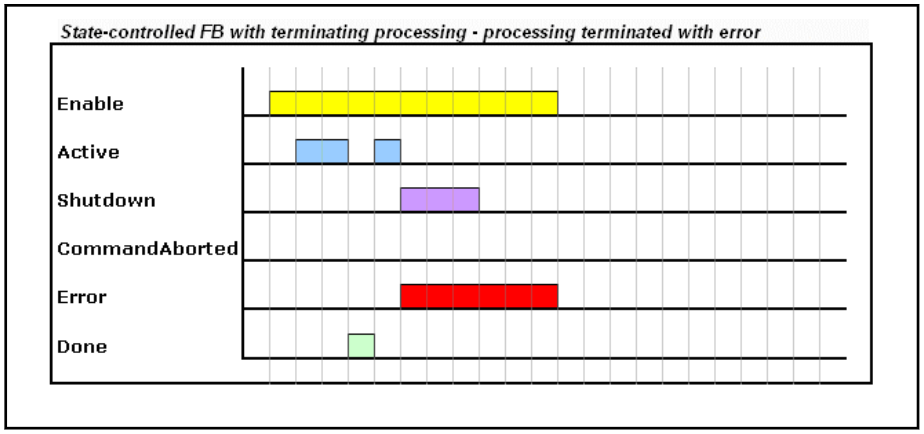


Fig. 2-5: I/O of state-controlled function blocks, final processing with "Shutdown". Processing completed with error

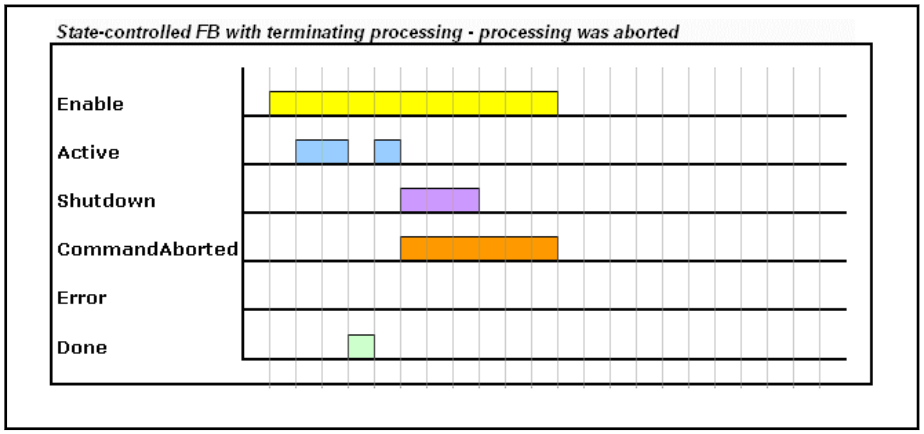


Fig. 2-6: I/O of state-controlled function blocks, final processing with "Shutdown". Processing was interrupted

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

Control/ processing	I/O	Name	Description
Edge-controlled/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).

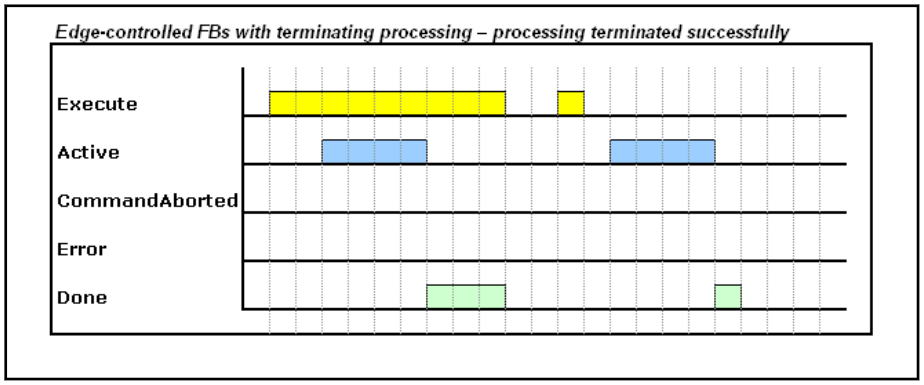


Fig. 2-7: I/O of edge-controlled function blocks, final processing without "Shutdown". Processing successfully completed

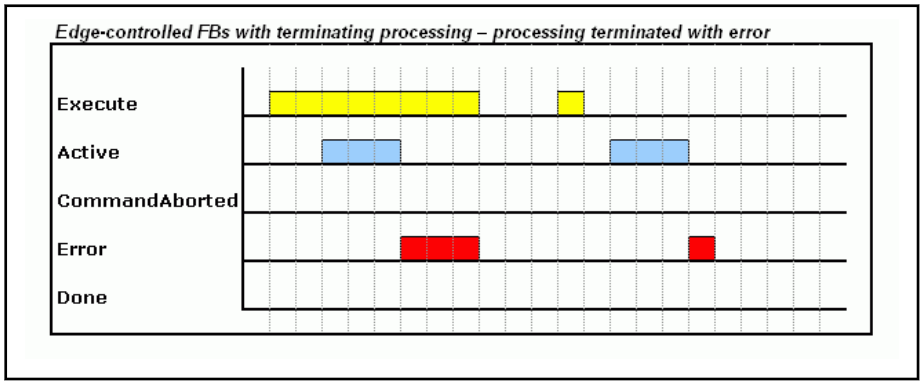


Fig. 2-8: I/O of edge-controlled function blocks, final processing without "Shutdown". Processing completed with error

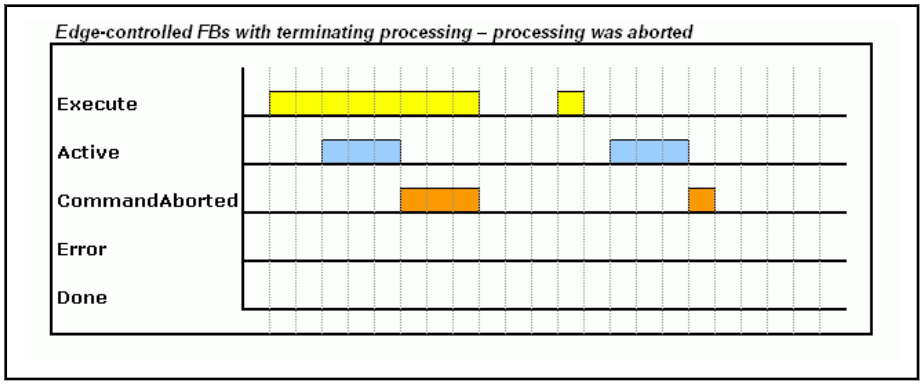


Fig. 2-9: I/O of edge-controlled function blocks, final processing without "Shutdown". Processing interrupted

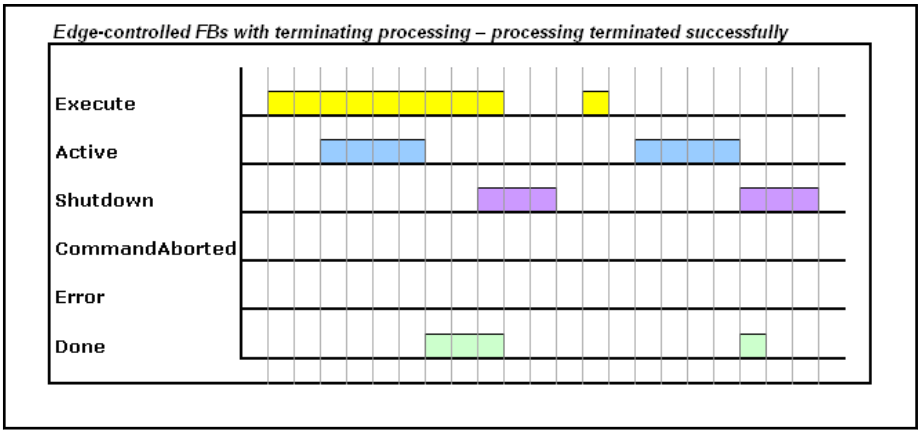


Fig. 2-10: I/O of edge-controlled function blocks, final processing with "Shutdown". Processing successfully completed

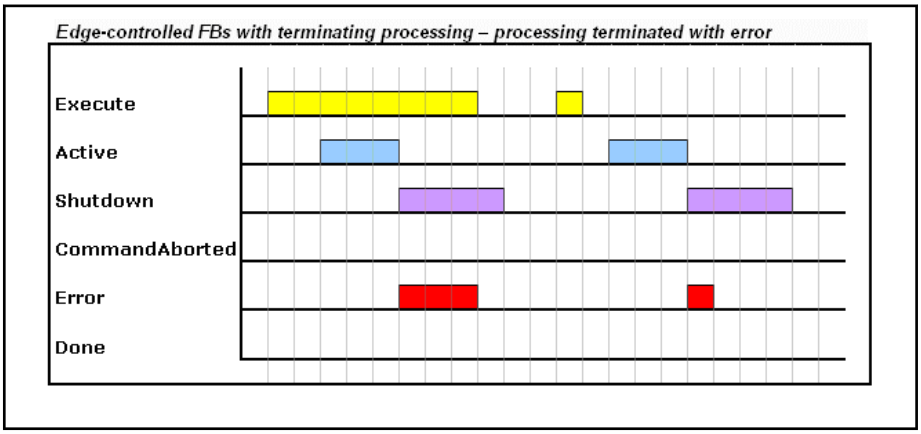


Fig. 2-11: I/O of edge-controlled function blocks, final processing with "Shutdown". Processing completed with error

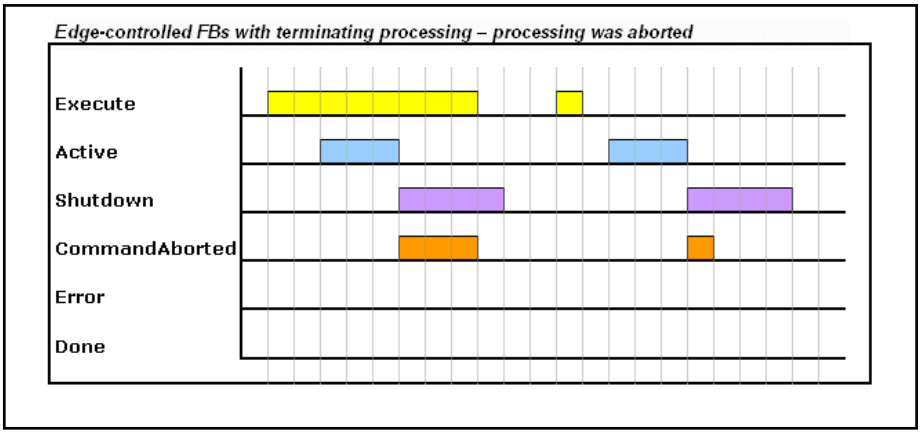


Fig. 2-12: I/O of edge-controlled function blocks, final processing with "Shutdown". Processing interrupted

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).

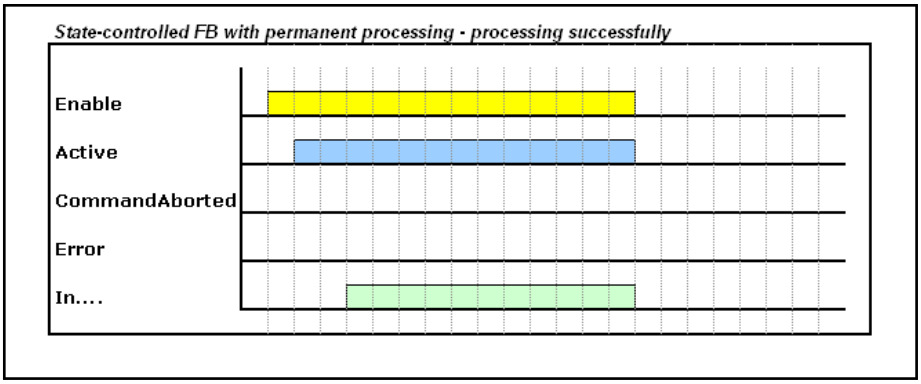


Fig. 2-13: I/O of state-controlled function blocks, permanent processing without "Shutdown". Processing successful

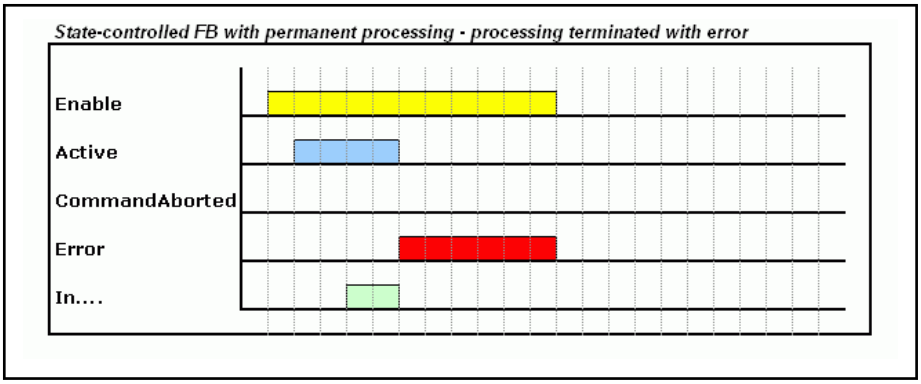


Fig. 2-14: I/O of state-controlled function blocks, permanent processing without "Shutdown". Processing aborted with errors

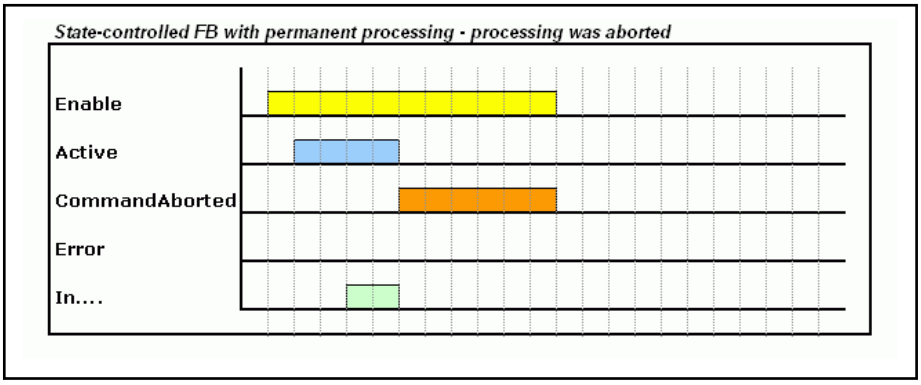


Fig. 2-15: I/O of state-controlled function blocks, permanent processing without "Shutdown". Processing interrupted

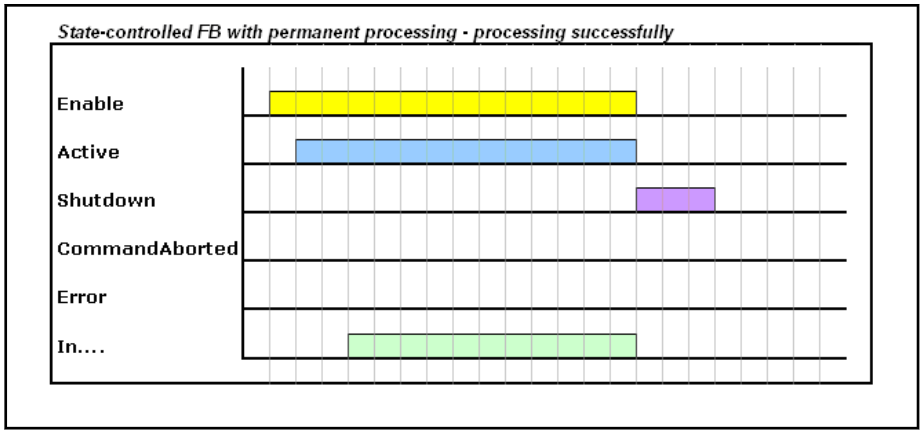


Fig. 2-16: I/O of state-controlled function blocks, permanent processing with "Shutdown". Processing successful

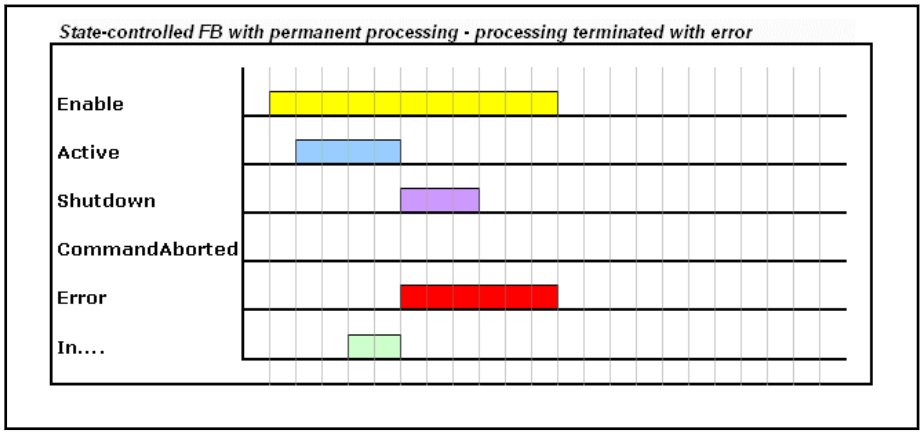


Fig. 2-17: I/O of state-controlled function blocks, permanent processing with "Shutdown". Processing aborted with errors

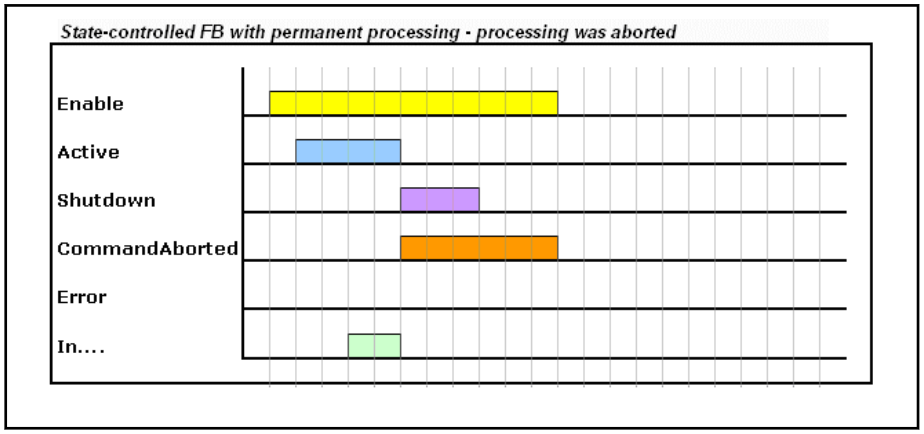


Fig. 2-18: I/O of state-controlled function blocks, permanent processing with "Shutdown". Processing interrupted

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 preprocessing 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).

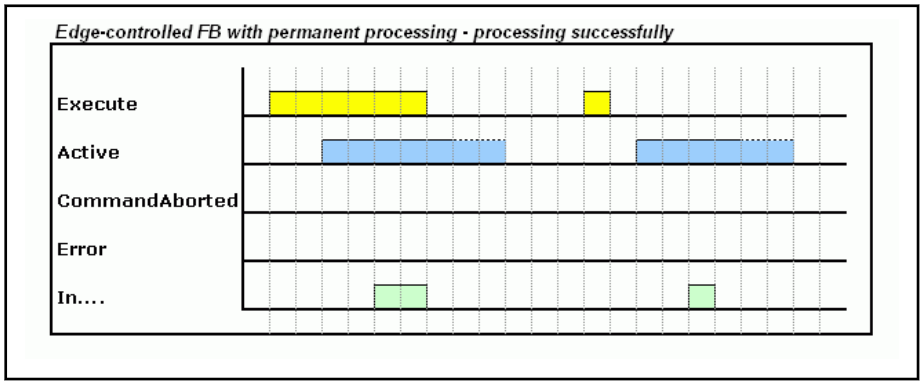


Fig. 2-19: I/O of edge-controlled function blocks, permanent processing without "Shutdown". Processing successful

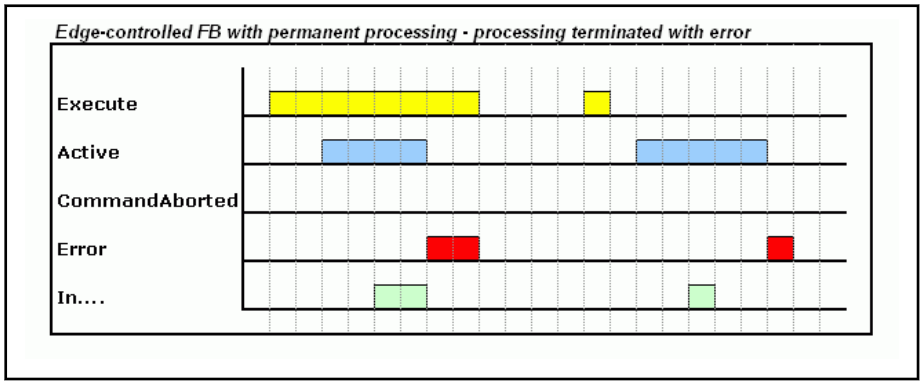


Fig. 2-20: I/O of edge-controlled function blocks, permanent processing without "Shutdown". Processing aborted with errors

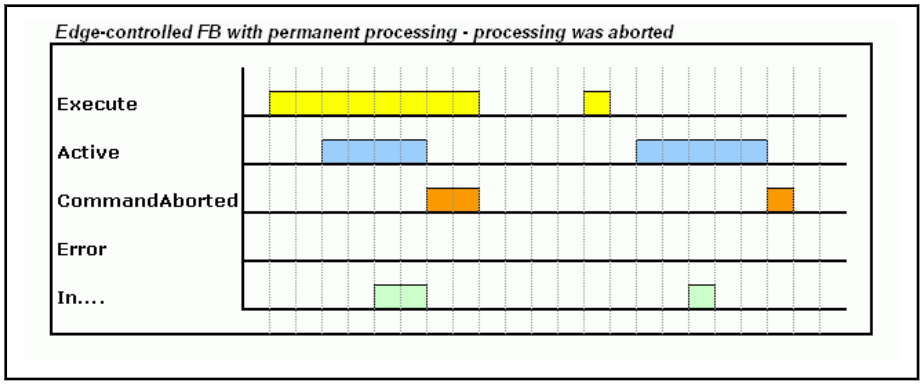


Fig. 2-21: I/O of edge-controlled function blocks, permanent processing without "Shutdown". Processing interrupted

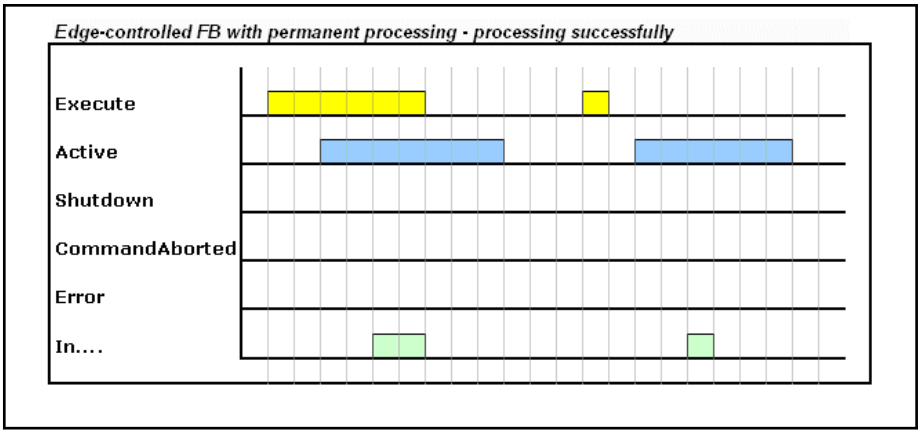


Fig. 2-22: I/O of edge-controlled function blocks, permanent processing with "Shutdown". Processing successful

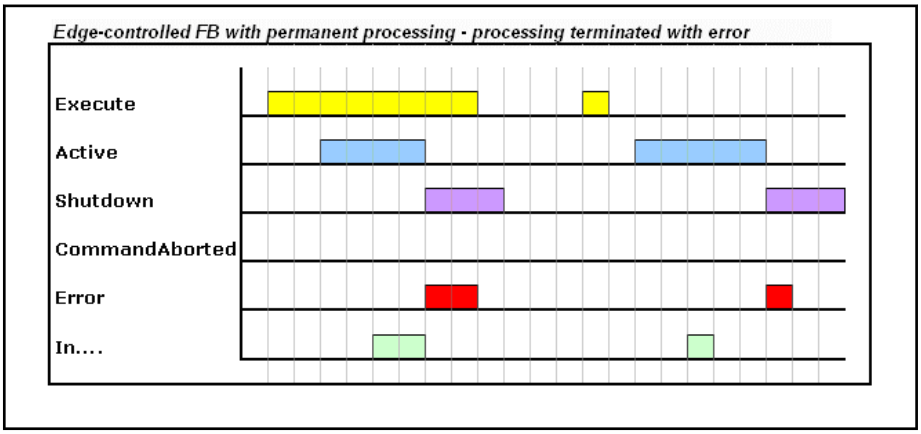


Fig. 2-23: I/O of edge-controlled function blocks, permanent processing with "Shutdown". Processing aborted with errors

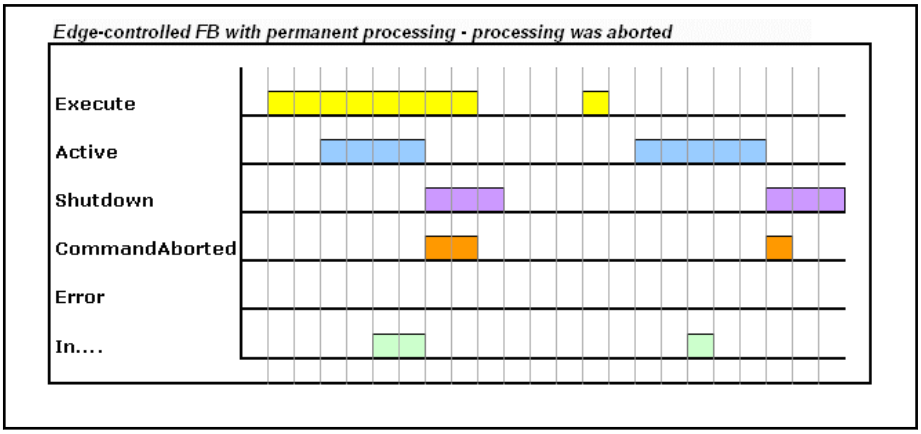


Fig. 2-24: I/O of edge-controlled function blocks, permanent processing with "Shutdown". Processing interrupted

3 RIL_ProfibusDP_02.library

3.1 Introduction and overview

History

When using the function module Profibus DP master and the existing onboard master, there are controls available to implement systems with several DP masters.

The RIL_ProfibusDP_02.library is provided. It can distinguish multiple DP masters.

This library is extended to version 01V02 to ensure compatibility.

☞ RIL_ProfibusDP.lib version 01V01 is not suitable for being on systems with several DP masters.

Areas of application and function blocks of RIL_ProfibusDP_02

Area of application (folder in library)	
Function blocks	**Description**
IL_DPMasterState, page 37	The function block determines the state of a local bus master as well as of the configured slaves
IL_DPBaudrateGet, page 48	Reads the current baud rate
IL_DPDeviceListGet, page 47	List of active DP devices in the system
IL_DPDevInfoGet, page 48	Reads the bus master information structure
IL_DPGetCfg, in preparation	
IL_DPIdent, page 39	Function to assemble Profibus IDs
IL_DPPrjSlaveListGet, page 49	List of configured slaves
IL_DPReadDiag, page 43	Reads the diagnostic data of a slave according to DP standard
IL_DPSlaveDiagListGet, page 50	List of current slave diagnostics
IL_DPSycFr, page 45	Synchronizes/freezes the control command
IL_DPV1Read, page 40	Reading V1 service
IL_DPV1Write, page 41	Writing V1 service

Tab. 3-1: RIL_ProfibusDP_02 library

3.2 Data types

3.2.1 IL_BUSMASTER

Brief description

The enumeration type IL_BUSMASTER of the RIL_FieldbusTypes.library is used to select the respective Profibus DP master. In addition to the onboard real-time Ethernet interface, other controllers can also be available in an MLC/MTX system via function modules.

The Profibus DP masters are distinguished on the basis of their ascending order in the configuration.

Indexes are assigned to the masters in the following order:

1. Onboard master
2. Function module 1
3. Function module 2

4. Function module 3
5. Function module 4

☞ If, for example, the onboard master is no Profibus master while a Profibus master is configured on function module 1, the value IL_BUSMASTER_0 is assigned to the master on function module 1.

Element	Value	Description
IL_BUSMASTER_0	0	First device
IL_BUSMASTER_1	1	Second device
IL_BUSMASTER_2	2	Third device
IL_BUSMASTER_3	3	Fourth device
IL_BUSMASTER_4	4	Fifth device

Tab. 3-2: IL_BUSMASTER enumeration type

Examples of addressing the various instances of the Profibus masters on a control:

Configured:	Addressed with:
Onboard: Profibus DP master	IL_BUSMASTER_0
FM1: Profibus DP slave	IL_BUSSLAVE_0
FM2: Profibus DP master	IL_BUSMASTER_1
FM3: Profibus DP master	IL_BUSMASTER_2

3.2.2 Slave Diagnostic Data according to Profibus DP Standard

Overview Structure of Profibus-specific diagnostic information according to the Profibus DP standard.

Slave diagnostic data are subdivided as follows:

- General part with a defined length of 6 bytes
- Extended diagnostics (slave-specific with variable length)

Offset	Type	Name	Description
0	BYTE	Stationsstatus_1	See below
1	BYTE	Stationsstatus_2	See below
2	BYTE	Stationsstatus_3	See below
3	BYTE	Master_Add	Bus address of the master having parameterized the slave
4	WORD	Ident_Number	Ident number of the slave
6-243		Ext_Diag_Data	Extended diagnostics according to Profibus DP standard

Tab. 3-3: Slave diagnostic data

Station state The following description of station states 1 to 3 is an excerpt from the Profibus DP standard.

Bit	Name	Description
7	Master_Lock	The DP slave has been parameterized by a different master. This bit is set by the DP master (class 1) if the address in octet 4 is unequal to 255 and unequal to its own address. The DP slave sets this bit permanently to zero
6	Prm_Fault	This bit is set by the DP slave if the last parameter telegram was invalid, e.g. incorrect length, incorrect Ident_Number, invalid parameters
5	Invalid_Slave_Response	This bit is set by the DP master as soon as an addressed DP slave receives an implausible answer. The DP slave sets this bit permanently to zero
4	Not_Supported	This bit is set by the DP slave as soon as a function was requested that is not supported by this DP slave
3	Ext_Diag	This bit is set by the DP slave. If the bit is set, a diagnostic entry has to be in the slave-specific diagnostic area (Ext_Diag_Data). If the bit is not set, a status message may be present in the slave-specific diagnostic area (Ext_Diag_Data). The meaning of this status message has to be defined according to the respective application
2	Cfg_Fault	This bit is set by the DP slave as soon as the configuration data previously received by the DP master does not correspond to the data determined by the DP slave
1	Station_Not_Ready	This bit is set by the DP slave if the DP slave is not yet ready for data exchange
0	Station_Non_Existent	This bit is set by the DP master if this DP slave cannot be reached via the bus. If this bit is set, the diagnostic bits contain the state of the previous diagnostic message or the initial value. The DP slave sets this bit permanently to zero

Tab. 3-4: Stationsstatus_1

Bit	Name	Description
7	Deactivated	This bit is set by the DP master as soon as the DP slave has been labeled as inactive in the DP slave parameter set and has been removed from cyclic processing. The DP slave always sets this bit to zero
6	Reserved	-
5	Sync_Mode	This bit is set by the DP slave as soon as it has received the Sync control command. A change in these bits does not cause a diagnostic message. That is why these bits usually do not reflect the current state
4	Freeze_Mode	This bit is set by the DP slave as soon as it has received the Freeze control command. A change in these bits does not cause a diagnostic message. That is why these bits usually do not reflect the current state
3	WD_On	Watchdog This bit is set by the DP slave as soon as its response monitoring is activated
2	1	This bit is set to "1" by the DP slave
1	Stat_Diag	Static diagnostics If the DP slave sets this bit, the DP master has to retrieve diagnostic information until this bit is cleared again. The DP slave sets this bit, for example, if it cannot provide any valid user data
0	Prm_Req	If the DP slaves sets this bit, the bit has to be parameterized and configured again. The bit remains set until parameterization has been completed. This bit is set by the DP slave

Tab. 3-5: Stationsstatus_2

Bit	Name	Description
7	Ext_Diag_Overflow	If this bit is set, the quantity of diagnostic information exceeds that specified in Ext_Diag_Data. The DP slave sets this bit, for example if the number of channel diagnostics exceeds that which the DP slave can enter in its transmitter buffer; or the DP master sets this bit if the DP slave sends more diagnostic information than the DP master is able to enter in its diagnostic buffer
6	Reserved	-
5	Reserved	-
4	Reserved	-
3	Reserved	-
2	Reserved	-
1	Reserved	-
0	Reserved	-

Tab. 3-6: Stationsstatus_3

3.2.3 Bus master status word

The bus master status word "BmState" provides an overview of the state of the bus master and the slaves at the field bus. It can be seen for example whether a slave diagnostics is present at least one bus master. Each bit that is set (TRUE) in "BmState" represents a status:

Bit	Status	Description
0	BMS_BMF	Bus master error: This bit indicates that there is a bus master error. In this case, the bus master error word contains more detailed information
1	BMS_KSD	Classified slave diagnostics: If this bit is set, at least one slave signals classified diagnostics. Bits 8 to 13 can be used to determine the classified diagnostics set
2	BMS_SD	Slave diagnostics: If this bit is set, at least one slave signals diagnostics
3	-	Reserved
4	-	Reserved
5	-	Reserved
6	-	Reserved
7	BMS_AKTIV	Active detection: This bit always has to be 1. If this is not the case, there is a fatal error in the software of the bus master
8	BMS_SNE	One or more slaves cannot be reached via the bus
9	BMS_SKF	One or more slaves signal configuration errors
10	BMS_DPS	One or more slaves signal static diagnostics
11	BMS_EXD	One or more slaves signal extended diagnostics
12	BMS_SNB	One or more slaves are not ready for cyclic data exchange
13	BMS_SF	One or more slaves signal a miscellaneous error

Bit	Status	Description
14	-	Reserved
15	-	Reserved

Tab. 3-7: Status coding in "BmState"

3.2.4 Bus master error word

The "BmError" bus master error word indicates serious errors which prevent operation at the field bus. Each bit that is set (TRUE) in "BmError" represents an error:

Bit	Error[1)]	Description
0	IL_BMF_HW_ERR	Hardware fault
1	IL_BMF_MPS_ERR	Master parameter set (field bus configuration file) missing or defective
2	IL_BMF_BUS_ERR	Error at field bus (e.g., short-circuit)
3	IL_BMF_SW_ERR	System error in periphery driver (i.e., the driver software has detected a serious error)

Tab. 3-8: Error coding in "BmError"

3.2.5 DP_SLAVELIST, array

Brief description

The bits list "DP_SLAVELIST" (DP_BITLIST) has a specified length of 16 bytes (128 bits).

Type declaration

```
TYPE  DP_SLAVELIST :
    ARRAY  [0..15] OF BYTE;
END_TYPE
```

Coding bit list

Each bit of the bit list is assigned to a bus address of the slave (Profibus: FDL address). The lowest-order bit in the first array element (ARRAY[0]) is assigned to the Profibus device with the address "0":

1) "IL_BMF_OK" indicates that there is no error

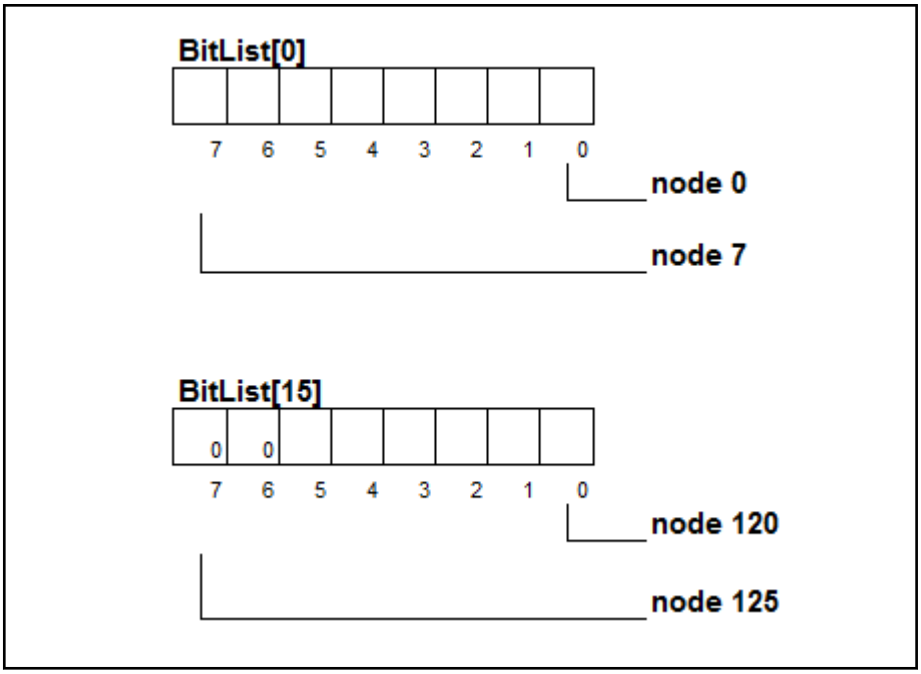


Fig. 3-1: Coding bit list

3.2.6 DP_DEVICELIST, structure

Brief description This data type summarizes information on a DP master:

Example *Program:*

```
TYPE  DP_DEVICELIST:
STRUCT
 bMasterAdr          :  BYTE;   (* master addressing *)
 bMasterBusAdr       :  BYTE;   (* master bus address *)
 wMasterError        :  WORD;   (* cf. IL_DPDevInfoGet *)
 wMasterState        :  WORD;   (* cf. IL_DPDevInfoGet *)
 udBaudrate          :  UDINT;  (* cf. IL_DPBaudrateGet *)
 dFirmwareVersion    :  DINT;   (* driver firmware version *)
 dHardwareVersion    :  DINT;   (* hardware version *)
 dAddInfo1           :  DINT;   (* 3S module identification *)
 dAddInfo2           :  DINT;   (* res *)
END_STRUCT
END_TYPE

TYPE  DP_MASTERLIST:
 ARRAY [0..5] of DP_DEVICELIST;  (* list for 6 masters *)
END_TYPE
```

An existing onboard slave is also listed.

3.2.7 DP_MASTERLIST, array

Brief description This data type summarizes information on up to 6 DP masters:

Example

```
TYPE  DP_DEVICELIST:
STRUCT
 bMasterAdr          :  BYTE;   (* master addressing *)
 bMasterBusAdr       :  BYTE;   (* master bus address *)
 wMasterError        :  WORD;   (* cf. IL_DPDevInfoGet *)
 wMasterState        :  WORD;   (* cf. IL_DPDevInfoGet *)
 udBaudrate          :  UDINT;  (* cf. IL_DPBaudrateGet *)
 dFirmwareVersion    :  DINT;   (* driver firmware version *)
 dHardwareVersion    :  DINT;   (* hardware version *)
 dAddInfo1           :  DINT;   (* 3S module identification *)
 dAddInfo2           :  DINT;   (* res *)
END_STRUCT
```

```
END_TYPE

TYPE  DP_MASTERLIST:
 ARRAY [0..5] of DP_DEVICELIST;  (* list for 6 masters *)
END_TYPE
```

3.2.8 DP_MASTERINFO, structure

Brief description This data type summarizes information on the bus master state.

Example

```
TYPE  DP_MASTERINFO: (*DP_DEVICEINFO:*)
STRUCT
    BmState    :  WORD;      (* bus master status *)
    BmError    :  WORD;      (* bus master error word *)
END_STRUCT
END_TYPE
```

3.3 Selecting DP master

Addressing The DP masters are distinguished based on their order in the DP configuration:

0 .. n (5) number of the DP master in ascending order of the configuration.

The only items counted are the DP masters.

Usage The functions and function blocks which have to access a certain instance of a master receive the "Master" input parameter of type BYTE.

The functions and function blocks which comply with the Profibus Guideline 2182 have an "ID" DWORD parameter which is interpreted as a slot handle. A master selection byte is reserved in this slot handle. The function **DP_SLOT** can be used to generate the parameter ID.

The functions and function blocks using the "Ident" DWORD parameter can generate this parameter using the function **IL_DPIdent**.

Byte	Contents	Description
0	MASTER	ID of DP system: Labeling of the DP master (or of the onboard slave)
1	SEGMENT	Number of the DP segment (0)
2	STATION	Number of the DP slave (bus address)
3	SLOT	Number of the slot in the slave

Tab. 3-9: Slot handle: Parameter ID

3.4 IL_DPMasterState

Brief description The **IL_DPMasterState** function block is used to determine the state of a local bus master and the configured slaves. The result is displayed as bit-coded in the "State" output parameter.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-10: Reference table

Interface description

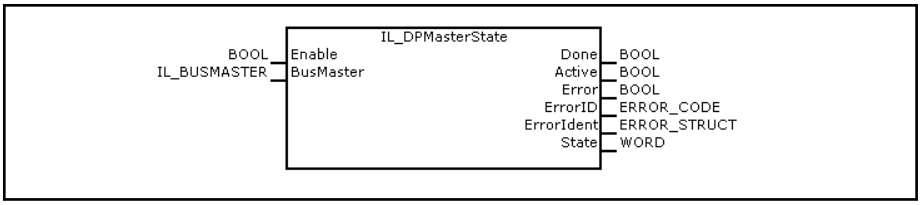


Fig. 3-2: Interface description of the IL_DPMasterState function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing enabled for function block (continuous, state-controlled)
	BusMaster	IL_BUSMASTER	Instance of the master in the order of the configuration tree. If only one Profibus DP master is available, the value of this master is always IL_BUSMASTER_0 Also refer to chapter 3.2.1 "IL_BUSMASTER" on page 31
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	State	WORD	Master diagnostic state (see elements of the State output)

Tab. 3-11: I/O interface of the IL_DPMasterState function block

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

"State" output (WORD)

Bit no.	Name	Description
0	IL_FBM_BUS_OFF	Field bus not ready
1	IL_FBM_MASTER_FAILURE	Bus master failure
2	IL_FBM_SLAVE_NOT_REACHABLE	One or more field bus slaves cannot be reached (slave not reachable)
3	IL_FBM_SLAVE_DIAGNOSIS	One or more field bus slaves report diagnoses (slave diag)
4	IL_FBM_SLAVE_ERROR	One or more field bus slaves report errors (slave error)

Tab. 3-12: Elements of "State" output

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	TRUE state of "Enable"

Tab. 3-13: Min./max. and default values of the IL_ DPMasterState inputs

Functional description

- The **IL_DPMasterState** is called and the parameter "BusMaster" is applied if Enable = TRUE
- If Enable = FALSE, the call does not have any effect. In this case, all output parameters are set to "0"

- Initially, the result of the call is shown in output parameters Done, Active and Error. If Active = FALSE, the call is completed and the other parameters have to be evaluated
- If Done = TRUE and Error = FALSE, no error occurred during the call
- If Done = FALSE and Error = TRUE, errors occurred during the call. In this case, the parameters ErrorID and ErrorIdent describe the error cause
- After the call has been completed successfully, the effective “State” output parameter is valid and can be evaluated by the calling instance
- If errors occurred, the effective “State” output parameter is invalid
- If the output parameter is valid, “State” shows the state of the bus master instance and the configured slaves in bit-encoded format
- State = 0 indicates that the state of the bus system is trouble-free
- If Error = TRUE, errors occurred during the call. In this case, ErrorID and ErrorIdent define the error cause. The output parameter “State” is invalid

Troubleshooting

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#1001	BusMaster instance	Invalid BusMaster instance (value range)
DEVICE_ERROR	16#2001	BusMaster instance	Bus master instance not available
ACCESS_ERROR	16#2002	ERROR_CODE	Error during access

Tab. 3-14: Error codes of function block IL_DPMasterState

3.5 IL_DPIdent

Brief description The **IL_DPIdent** function creates an Ident handle from individual components.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-15: Reference table

Interface description

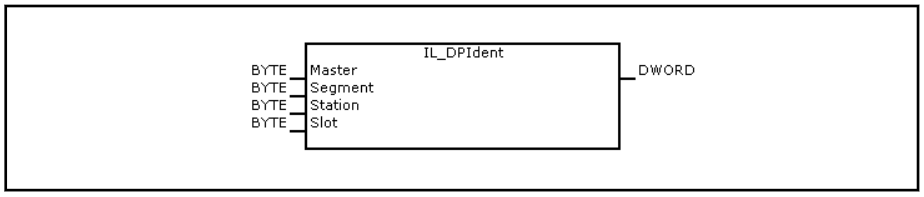


Fig. 3-3: Interface description of the IL_DPIdent function

I/O type	Name	Type	Description
VAR_INPUT	Master	BYTE	DP master identification (cf. above)
	Segment	BYTE	Number of the DP segment (0)
	Station	BYTE	Number of the DP slave (bus address): If the command is to apply for only one special slave, the bus address of the slave has to be entered here (0...125). Only for IL_DPSycFr: However, if the command is to be entered for all slaves of a group, enter the global address (= 127) here
	Slot	BYTE	Number of the slot in the slave (acc. to the slave specification), value range: 0...254
Function value		DWORD	Ident handle

Tab. 3-16: I/O interface of the IL_DPIdent function

Functional description

The DWORD "Ident" is created from the four byte values. This DWORD is required as input parameter for the function blocks described in the following.

3.6 IL_DPV1Read

Brief description

Read-only DPV1 access with the function block **IL_DPV1Read**. The data exchange on the Profibus DP is executed acyclically. For the process data to be read, a target area has to be defined via a pointer addressing (POINTER).

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-17: Reference table

Interface description

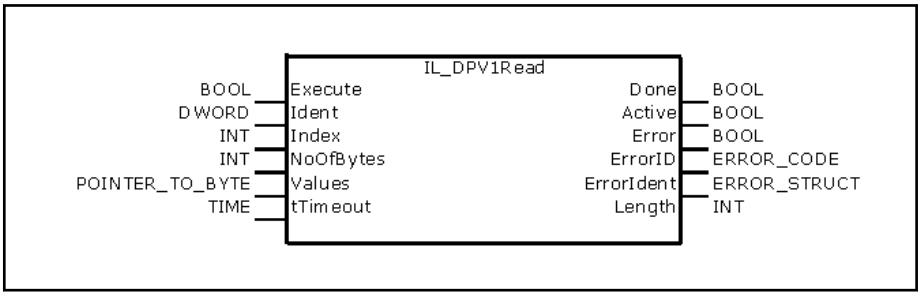


Fig. 3-4: Interface description of the IL_DPV1Read function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Enabling the function; an enabled function block cannot be interrupted
	Ident	DWORD	Ident handle (refer to the function chapter 3.5 "IL_DPIdent" on page 39)
	Index	INT	Index of process data (array number)
	NoOfBytes	INT	Maximum length of the data to be read. Number of available bytes on the "Values" pointer
	Values	POINTER TO BYTE	Pointer to data buffer for target data
	tTimeout	TIME	Timeout for waiting for data

I/O type	Name	Type	Description
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table
	Length	INT	Length of the read data in bytes

Tab. 3-18: I/O interface of the IL_DPV1Read function block

Signal-time diagram

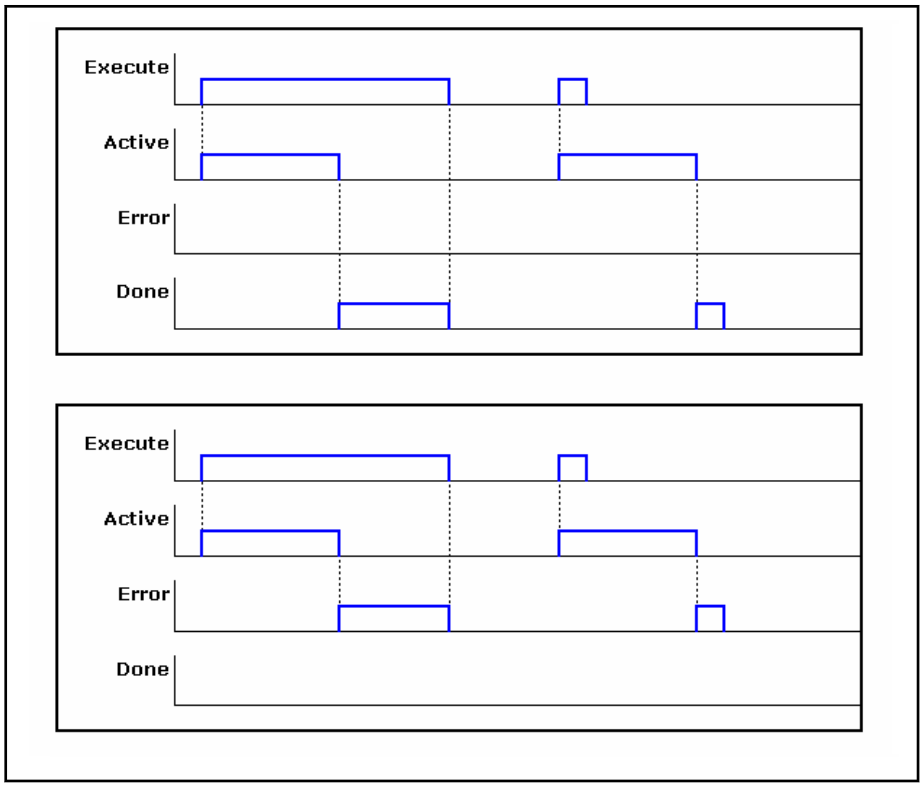


Fig. 3-5: Signal-time diagram IL_DPV1Read

Functional description

The master (class 1) accesses a DP-V1 slave. The master retrieves the data set of the slave addressed by "slave address", "slot" and "index". Addressing with "Slot" and "Index" as well as data interpretation is slave-specific. Thus, refer to the respective slave documentation. The execution time of the function also depends on the bus load and the set baud rate.

This function is only for slaves participating in the DP bus cycle.

3.7 IL_DPV1Write

Brief description

Write DPV1 access with the function block **IL_DPV1Write**. The data exchange on the Profibus DP is executed acyclically. The process data to be written are transfer using pointer addressing (a POINTER).

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-19: Reference table

Interface description

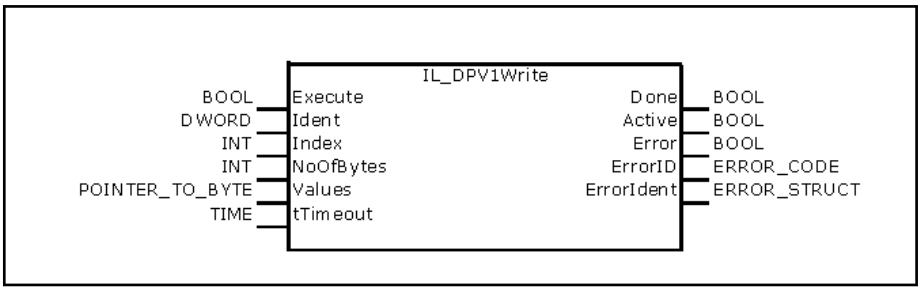


Fig. 3-6: Interface description of the IL_DPV1Write function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Enabling the function; an enabled function block cannot be interrupted.
	Ident	DWORD	Ident handle (see function IL_DPIdent)
	Index	INT	Index of process data (array number)
	NoOfBytes	INT	Maximum length of the data to be written. Number of available bytes on the "Values" pointer
	Values	POINTER TO BYTE	Pointer to data buffer for data
	tTimeout	TIME	Timeout for waiting for writing completion
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table

Tab. 3-20: I/O interface of the IL_DPV1Write function block

Signal-time diagram

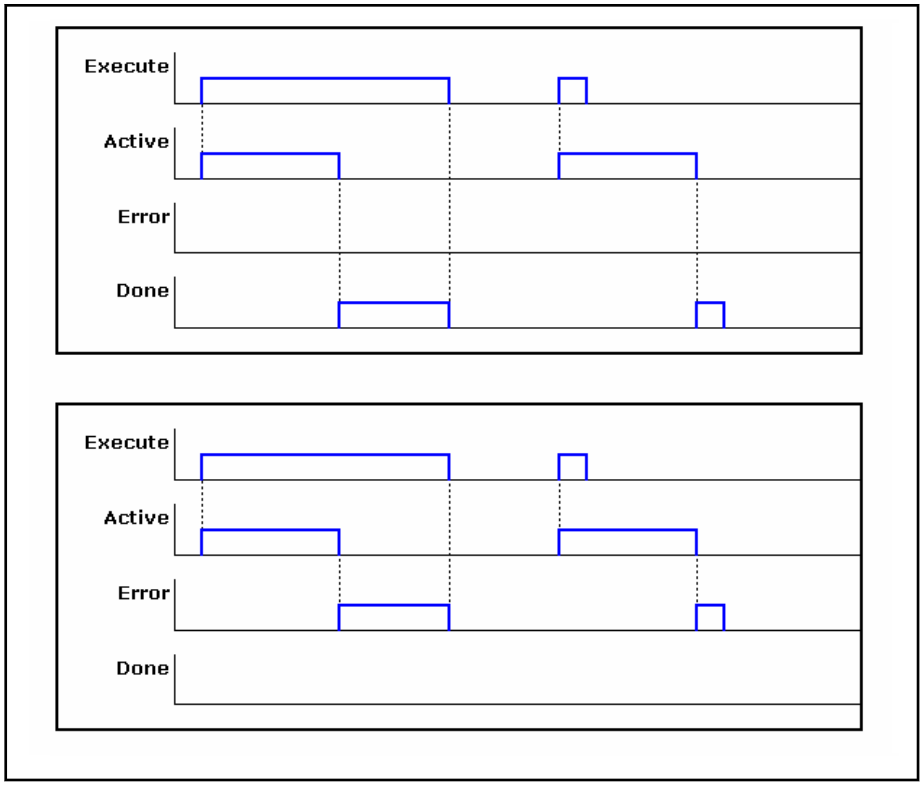


Fig. 3-7: Signal-time diagram IL_DPV1Write

Functional description

The master (class 1) accesses a DP-V1 slave. It retrieves the data set of the slave addressed with slave address, slot and index. Addressing with slot and index as well as data interpretation is slave-specific. Thus, refer to the respective slave documentation. The execution time of the function also depends on the bus load and the set baud rate.

This function is only for slaves participating in the DP bus cycle.

3.8 IL_DPReadDiag

Brief description

The **IL_DPReadDiag** function block retrieves the diagnostic data of a slave from the DP master (DPM1). The diagnostic data buffer has to be provided to address it via a pointer.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-21: Reference table

Interface description

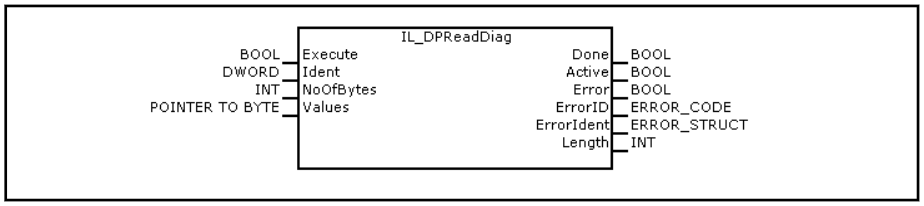


Fig. 3-8: Interface description of the IL_DPReadDiag function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Enabling the function; an enabled function block cannot be interrupted
	Ident	DWORD	Ident handle (see function IL_DPIdent)
	NoOfBytes	INT	Maximum length of the data to be read. Number of available bytes on the "Values" pointer
	Values	POINTER TO BYTE	Pointer to data buffer of slave diagnostic data. Refer to chapter 3.2.2 "Slave Diagnostic Data according to Profibus DP Standard" on page 32
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table
	Length	INT	Length of diagnostic data in bytes

Tab. 3-22: I/O interface of the IL_DPReadDiag function block

Signal-time diagram

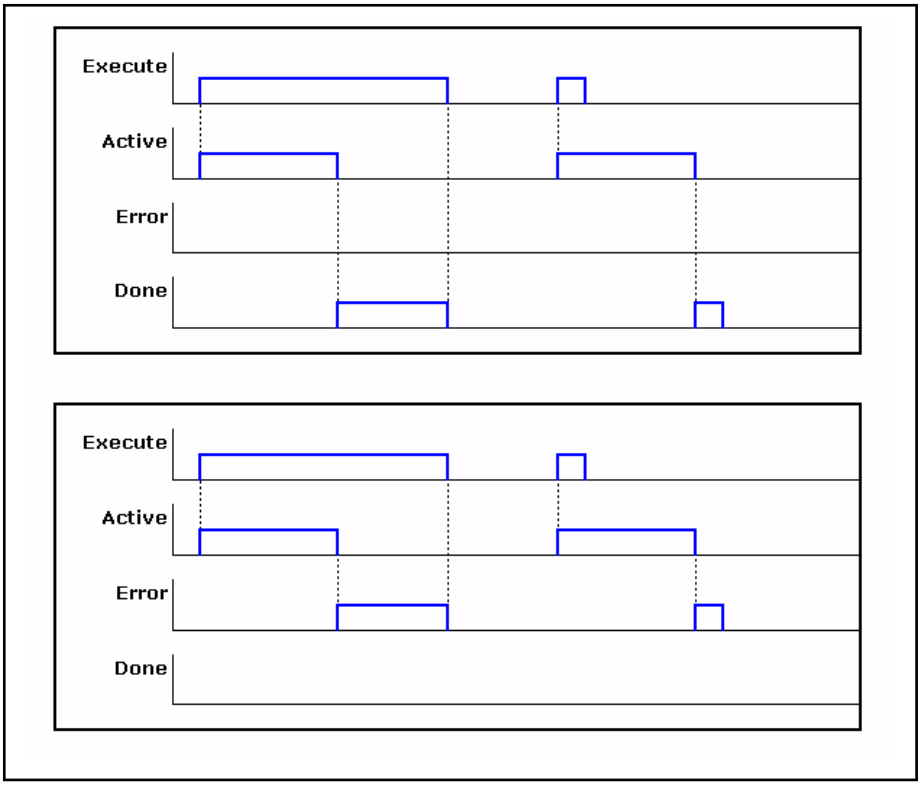


Fig. 3-9: Signal-time diagram IL_DPReadDiag

Functional description

The master retrieves the diagnostic information of the slaves and saves them slave-specifically. The slave triggers the diagnostic request of the master using request bits in the cyclic telegram traffic. The slave is responsible for the content of the diagnostic data. The present function does not initiate any telegram traffic at the Profibus. It only accesses the diagnostic information present in the master.

3.9 IL_DPSycFr

Brief description

Control commands to synchronize inputs and outputs can be implemented with the **IL_DPSycFr** function block.

Profibus DP allows the master to send a "Global Control Telegram" to a group of slaves. The "Global Control Telegram" contains the control commands "Freeze" and "Sync":

- With the **Freeze** control command, all slaves in the addressed group save the current **input data** simultaneously (synchronize inputs)
- With the **Sync** control command, all slaves in the addressed group accept the current **output data** simultaneously (synchronize outputs)

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-23: Reference table

Interface description

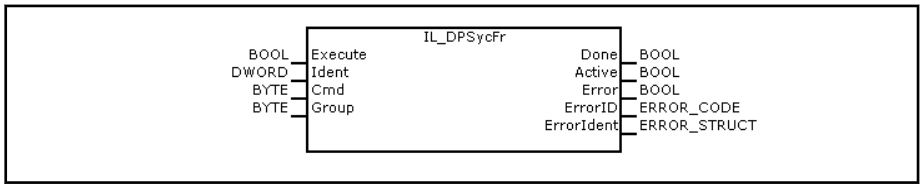


Fig. 3-10: Interface description of the IL_DPSycFr function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Enabling the function; an enabled function block cannot be interrupted
	Ident	DWORD	Ident handle, see IL_DPIdent (slot not relevant. Should be "0")
	Cmd	BYTE	Control command (see below)
	Group	BYTE	Selects one or multiple groups on which the command refers to. Each bit is assigned to a group
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table

Tab. 3-24: I/O interface of the IL_DPSycFr function block

Signal-time diagram

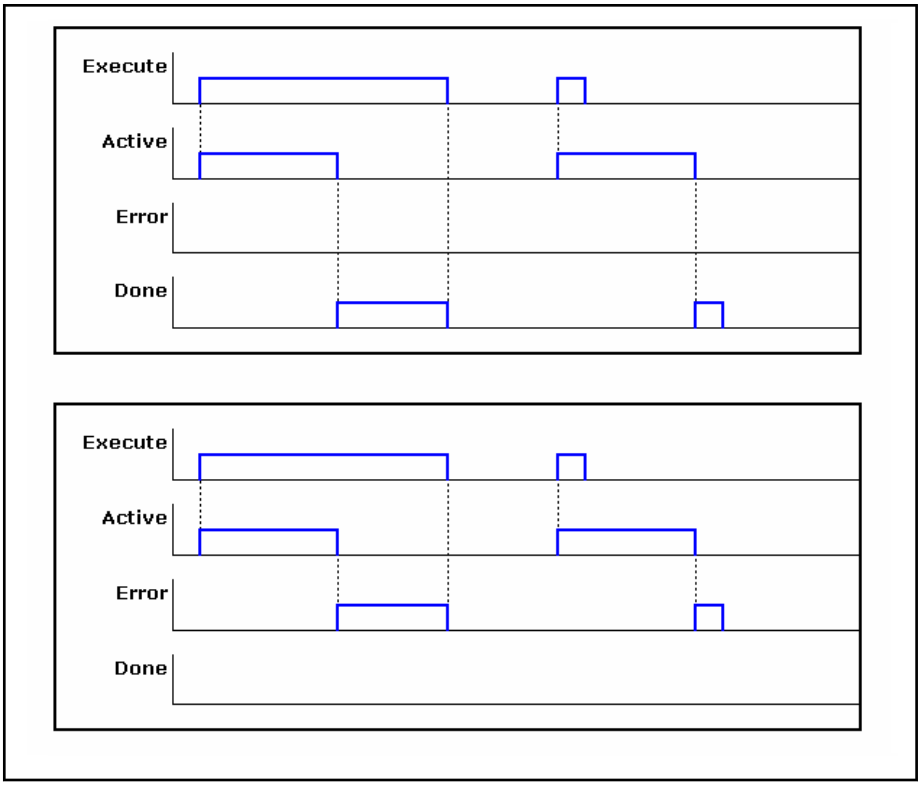


Fig. 3-11: Signal-time diagram IL_DPSycFr

Functional description

Example: To address all slaves of a group, the ID value is 16#007f0000.

To generate the ID from the individual components, use the **IL_DPIdent** function.



To synchronize the outputs, ensure that all slaves received the current output data, before receiving the Sync command.

Call **IL_DPSycFr** function block from the same PLC task used to write the output data of the slaves:

At first, the output data for the slaves to be synchronized is written in a PLC task. Subsequently, use **IL_DPSycFr** to start the Sync command. If the Sync command is not yet complete (BUSY), the output data may not be changed anymore.

When using Sync or Freeze, assign the groups at the master and the corresponding slaves in the IndraWorks Project Explorer. For more information, refer to the documentation or the IndraWorks online help.

Possible control commands

IL_DP_CMD_UNFREEZE	16#04	Freeze mode release
IL_DP_CMD_FREEZE	16#08	command Freeze
IL_DP_CMD_UNSYNC	16#10	Sync mode release
IL_DP_CMD_SYNC	16#20	command Sync

Tab. 3-25: Possible IL_DPSycF control commands

3.10 IL_DPDeviceListGet

Brief description

The **IL_DPDeviceListGet** function block determines a list of the currently available DP masters.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-26: Reference table

Interface description

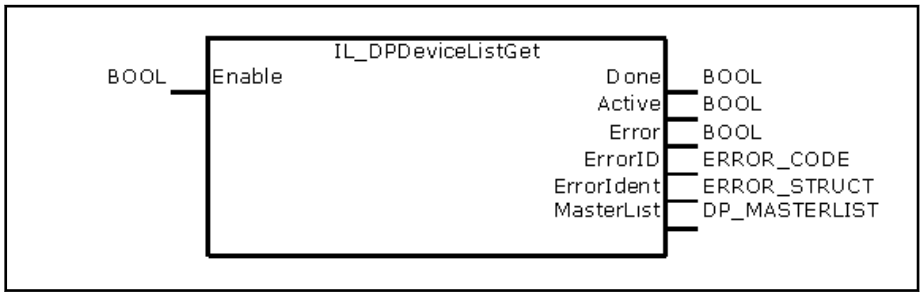


Fig. 3-12: Interface description of the IL_DPDeviceListGet function block

I/O type	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function enabled
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table
	MasterList	DP_MASTERLIST	List of available (up to 6) DP masters

Tab. 3-27: I/O interface of the IL_DPDeviceListGet function block

Signal-time diagram

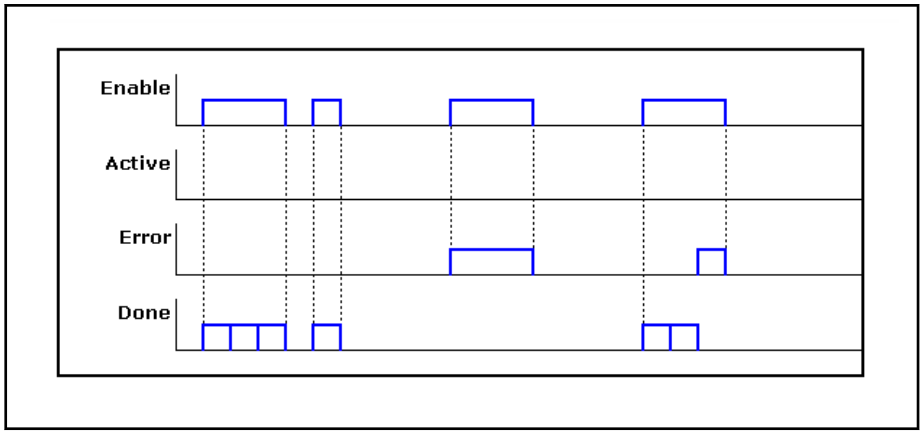


Fig. 3-13: Signal-time diagram IL_DPDeviceListGet

Functional description

This function block is used to indicate the active and addressable master instances on the PLC. The user can thus also monitor whether an appropriate configuration was loaded and correctly detected.

3.11 IL_DPBaudrateGet

Brief description

The **IL_DPBaudrateGet** function block determines the baud rate of the connected field bus. The baud rate is specified in bits/s.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-28: Reference table

Interface description

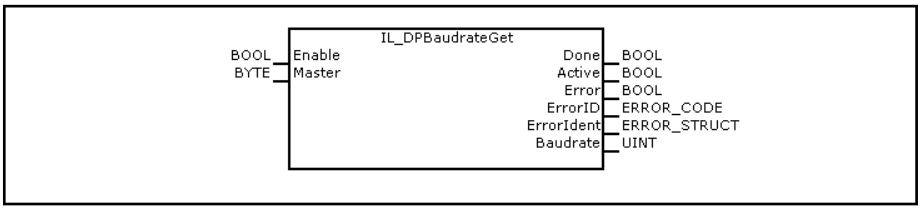


Fig. 3-14: Interface description of the IL_DPBaudrateGet function block

I/O type	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function enabled
	Master	BYTE	MasterID (see above)
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table
	Baud rate	UDINT	Baud rate in 1/s

Tab. 3-29: I/O interface of the IL_DPBaudrateGet function block

Signal-time diagram

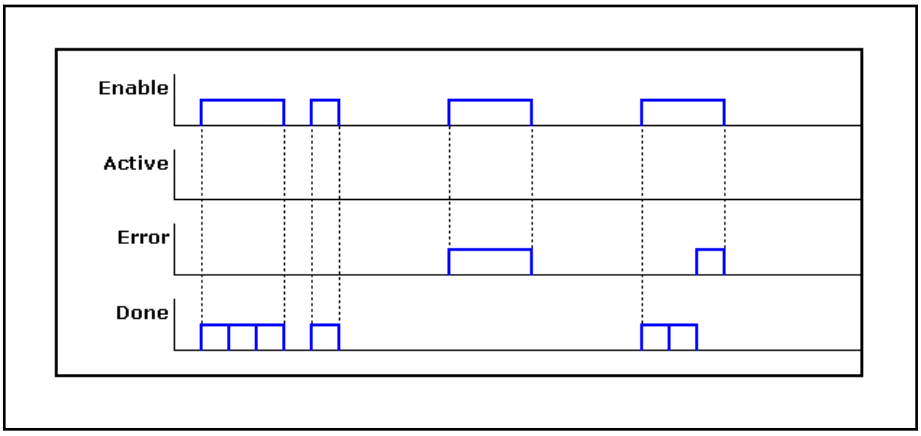


Fig. 3-15: Signal-time diagram IL_DPBaudrateGet

Functional description

Determining the operating state of the addressed PB master. Code "BmState" and "BmError" with 1.3.2 and 1.3.3.

3.12 IL_DPDevInfoGet

Brief description

The **IL_DPDevInfoGet** function block retrieves information on the operating state of the respective Profibus DP device.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-30: Reference table

Interface description

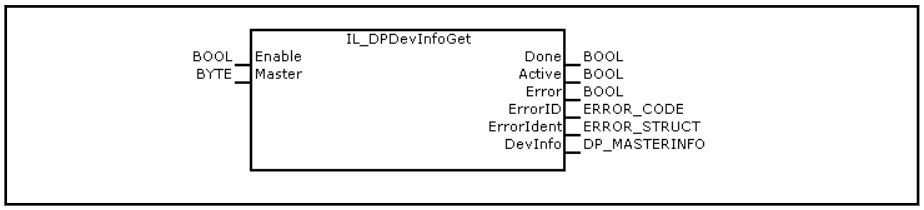


Fig. 3-16: Interface description of the IL_DPDevInfoGet function block

I/O type	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function enabled
	Master	BYTE	MasterID (see above)
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table
	DevInfo	DP_MASTERINFO	Contains information on the bus master state

Tab. 3-31: I/O interface of the IL_DPDevInfoGet function block

Signal-time diagram

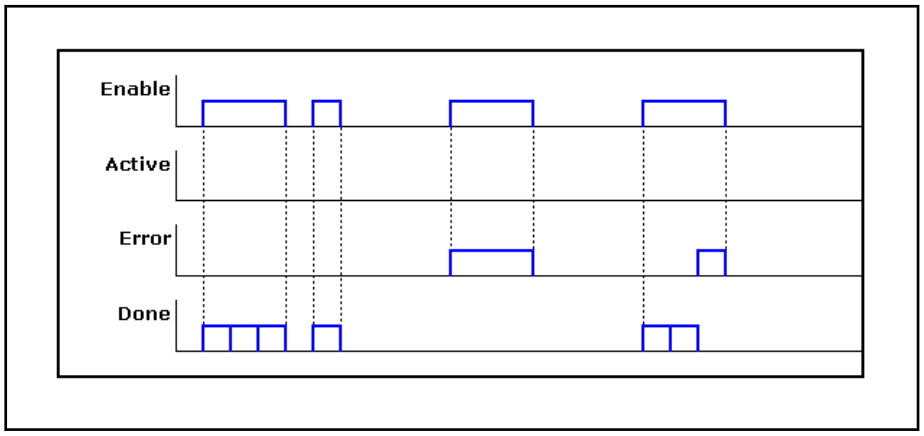


Fig. 3-17: Signal-time diagram IL_DPDevInfoGet

3.13 IL_DPPrjSlaveListGet

Brief description

The **IL_DPPrjSlaveListGet** function block provides the list of configured slaves. The list contains all slaves in the master configuration file.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-32: Reference table

Interface description

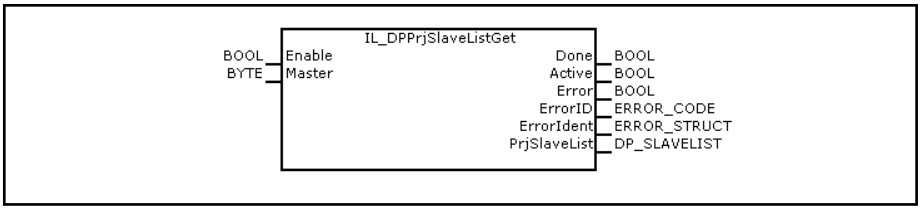


Fig. 3-18: Interface description of the IL_DPPrjSlaveListGet function block

I/O type	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function enabled
	Master	BYTE	MasterID (see above)
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table
	PrjSlaveList	DP_SLAVELIST	Bit list with a bit set for each slave reporting diagnostics Refer to chapter 3.2.5 "DP_SLAVELIST, array " on page 35

Tab. 3-33: I/O interface of the IL_DPPrjSlaveListGet function block

Signal-time diagram

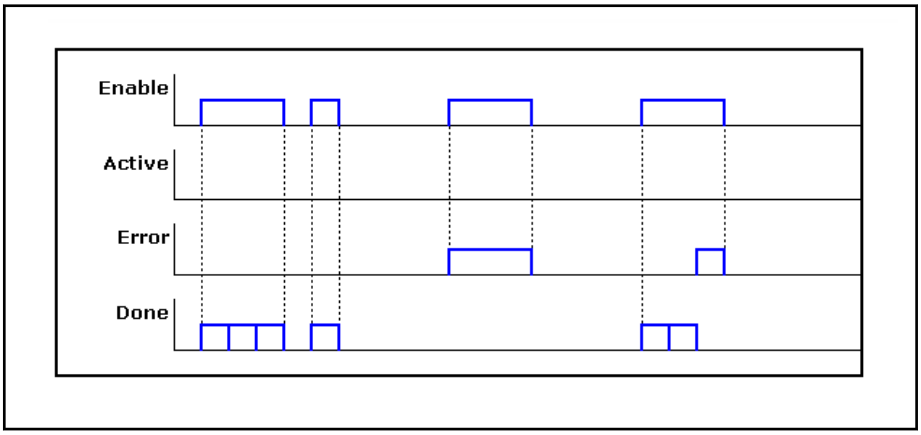


Fig. 3-19: Signal-time diagram IL_DPPrjSlaveListGet

Functional description

One bit is set in the bit list for each configured slave of a master. The function block is based on the existing configuration data.

3.14 IL_DPSlaveDiagListGet

Brief description

The **IL_DPSlaveDiagListGet** function block provides the list of slaves reporting a diagnostics.

Target system	Library/folder
MLC	RIL_ProfibusDP_02
MTX	RIL_ProfibusDP_02

Tab. 3-34: Reference table

Interface description

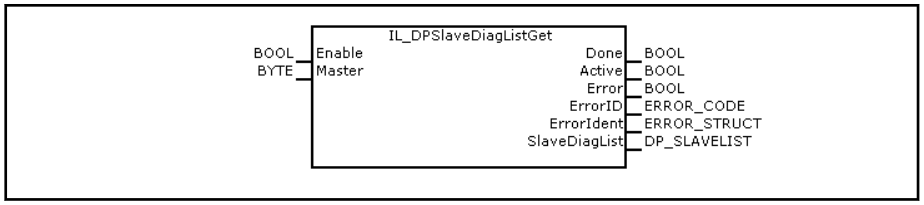


Fig. 3-20: Interface description of the IL_DPSlaveDiagListGet function block

I/O type	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function enabled
	Master	BYTE	MasterID (see above)
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Completed message (unsuccessful)
	ErrorID	ERROR_CODE	Refer to chapter 3.15.1 "ErrorID / ERROR_CODE" on page 51
	ErrorIdent	ERROR_STRUCT	Error structure from the PB_DP_TABLE error table
	SlaveDiagList	DP_SLAVELIST	Bit list with a bit set for each slave reporting diagnostics Refer to chapter 3.2.5 "DP_SLAVELIST, array " on page 35

Tab. 3-35: I/O interface of the IL_DPSlaveDiagListGet function block

Signal-time diagram

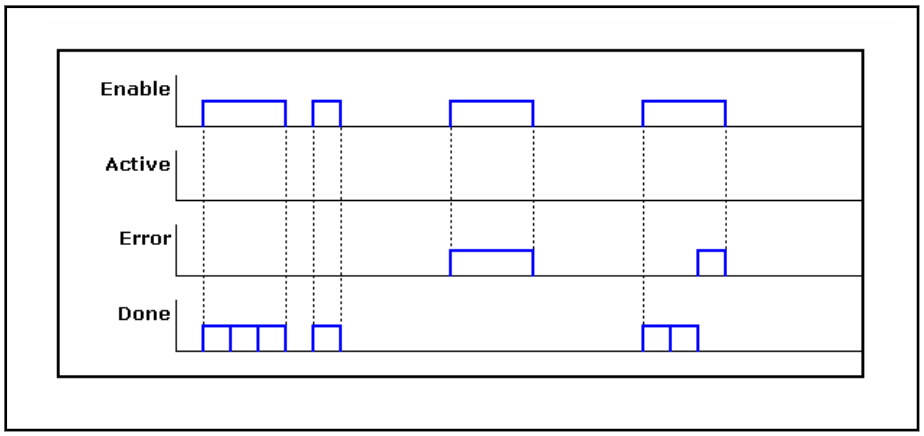


Fig. 3-21: Signal-time diagram IL_DPSlaveDiagListGet

Functional description

One bit is set in the "SlaveDiagList" for each diagnostic-reporting slave. Thus, diagnostics can be specifically read. No DP services are initiated by this function block. Data in the master is evaluated.

3.15 Error IDs

3.15.1 ErrorID / ERROR_CODE

Library	Range
RIL_CommonTypes.lib	Data types of POU diagnostics - including

Tab. 3-36: ERROR_CODE, defined in RIL_CommonTypes

Enumerator	Code	Description
NONE_ERROR	16#0000	No error code available
INPUT_INVALID_ERROR	16#0001	Invalid input assignment
COMMUNICATION_ERROR	16#0002	Communication error
RESOURCE_ERROR	16#0003	Source not available
ACCESS_ERROR	16#0004	Incorrect or invalid data access
STATE_MACHINE_ERROR	16#0005	State machine has invalid value
INPUT_RANGE_ERROR	16#0006	Value of one or multiple inputs is outside the defined limits
CALCULATION_ERROR	16#0007	Computation error
DEVICE_ERROR	16#0008	Drive error
OTHER_ERROR	16#7FFE	Undefined error (cannot be assigned to any other ID)
SYSTEM_ERROR	16#7FFF	System error

Tab. 3-37: Possible error IDs

3.15.2 ErrorIdent

Structure ErrorIdent is a structure of three elements. The structure is assigned with "0" by default.

- Error_Table: PB_DP_TABLE (16#0130) Profibus DP error
- ErrorAdditional1
- ErrorAdditional2

ErrorAdditional1 In ErrorAddtional1, a primary differentiation is made according to the error source. The following error sources are distinguished:

16#0001: Onboard Profibus Device

16#0002: FunctionModule (FM) Profibus Device

16#0003: netXDevice

16#0005: Specific error. For more information, refer to the Profibus DP standard chapter 3.15.6 "Error_Code_DP" on page 55 or to the device-specific documentation

16#0100: Function not supported

16#0101: Device (master) not found

16#0102: Slave not configured

16#0103: I/O driver not ready

16#0104: Timeout error

16#0107: Parameter error

ErrorAdditional2 ErrorAdditional2 is also divided into 4 bytes for "Onboard Profibus Device".

The bytes have the following significance:

Byte no.	Meaning	Description
Byte 3	Error_Source	Error origin is used for differentiations: 16#00 Profibus (slave) 16#10 master stack 16#20 Profibus FDL layer
Byte 2	Error_Code_DP Error_Code_OB Error_Code_FDL	For Error_Source = 16#00 For Error_Source = 16#10 For Error_Source = 16#20
Byte 1	AddInfo_1	Reserved
Byte 0	AddInfo_2	Reserved

Tab. 3-38: Coding the onboard Profibus

ErrorAdditional2 is also divided into 4 bytes for "FunctionModule (FM) Profibus Device".

The bytes have the following significance:

Byte no.	Meaning	Description
Byte 3	FM_Error_Code	Error code of the FM master
Byte 2	Error_Code_DP	Refer to the following tables
Byte 1	Error_Code_1	DP user-specific
Byte 0	AddInfo_1	Reserved

Tab. 3-39: Coding the function module

3.15.3 FM_Error_Code

The following error description corresponds to the "Error Response Definitions" of the Hilscher Profibus DP master.

FM_Error_Code	Error
16#02	No memory or buffer is available for this service on the slave
16#03	The slave does not support any DPV1 services
16#09	Slave did not send data
16#11	The slave did not respond/is not at the bus
16#12	DP master is not at the Profibus (check wiring)
16#19	Slave does not operate DPV1-compliant
16#36	Access was denied by the slave. Analyze Error_Code_DP!
16#81	DPV1 is not configured on master
16#82	The slave responded with implausible parameters
16#83	One service is also being processed. Parallel services not permitted
16#84	Data size exceeds configured size
16#85	Incorrect parameters in request

16#9a	Unknown command
16#F0	Invalid state

Tab. 3-40: Function module error code

3.15.4 Error_Code_OB

Error_Code_OB	Error
16#11	Invalid job parameters
16#23	RequestList full
16#25	Error at SemTake
16#31	Prohibited call
16#32	Invalid call parameter
16#33	Invalid data length
16#34	Incorrect call state
16#35	Slave not configured
16#36	Slave configured, but not operated cyclically
16#61	DPV1 request to non-DPV1 slave
16#62	Slave does not respond within timeout
16#63	DPV1 telegram format error
16#64	Task was withdrawn
16#65	Corresponding RQB not found
16#66	Invalid parameters
16#67	Unknown AMPRO2 Opcode

Tab. 3-41: Onboard error code

3.15.5 Error_Code_FDL

Error_Code_FDL	Error
16#61	FE: Format error in a request APDU
16#62	NI: Service not implemented
16#63	AD: access denied
16#64	EA: Area to large (upload/download)
16#65	LE: Data block length too large (upload/download)
16#66	RE: Format error in a response APDU
16#67	IP: Invalid parameter
16#68	SC: Sequence conflict
16#69	SE: Sequence error
16#6A	NE: Area non-existent
16#60	No slave found

Error_Code_FDL	Error
16#6B	DI: Data incomplete
16#6C	NC: Master parameter set not compatible

Tab. 3-42: Field bus data link layers (FDL) error code

3.15.6 Error_Code_DP

Structure

The meaning of "Error_Code_DP" corresponds to the "Error_Code_1" from the DPV1 standard. The bits 4..7 of the error byte generate the "Error_Class", the bits 0..3 generate the "Error_Code".

7	6	5	4	3	2	1	0	Meaning
				———				Error code
———								Error class

Tab. 3-43: Structure of the Error_Code_DP

Error_Class	Meaning	Error_Code
0 to 9	reserved[1)]	
10	Application	0 = read error 1 = write error 2 = module failure 3 to 7 = reserved[1)] 8 = version conflict 9 = feature not supported 10 to 15 = user-specific
11	Access	0 = invalid index 1 = write length error 2 = invalid slot 3 = type conflict 4 = invalid area 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10 to 15 = user-specific

Error_Class	Meaning	Error_Code
12	resource	0 = read constrain conflict 1 = write constrain conflict 2 = resource busy 3 = resource unavailable 4 to 7 = reserved[1)] 8 to 15 = user specific
13 to 15	User specific	

1) reserved values are intended to be passed unchanged to the user

Tab. 3-44: DP error code

Additional info: User- (slave-)specific

4 RIL_ProfibusDPSlave.library

4.1 Introduction and overview

The functionality described implements the acyclic READ and WRITE DPV1 services for Profibus DP slaves. This allows access to data objects at slave application level.

To implement the READ service, the function requires combined use of two function blocks (FBs).

FB IL_PBDPSlaveDPV1Polling is used to cyclically poll the activity of a DPV1 service request. If active, a "READ" request is checked by the user program for permissibility and answered by calling FB IL_PBDPSlaveDPV1Response.

The particular data object is accessed such that the parameters transferred to FB IL_PBDPSlaveDPV1Response are a pointer to the data object and the valid data length.

The data is copied by the FB.

A "WRITE" service requires combined use of three function blocks.

FB IL_PBDPSlaveDPV1Polling is used to cyclically poll the activity of a DPV1 service request. If active, a "WRITE" request is checked by the user program for permissibility. If necessary, the user data is copied to the target object by calling the IL_PBDPSlaveDPV1GetWriteData function block. To achieve this, a pointer to the data object and the valid data length are transferred as parameters to the IL_PBDPSlaveDPV1GetWriteData function block.

The data is copied by the FB.

The response is again given by calling FB IL_PBDPSlaveDPV1Response.

The maximum data length per access is 240 bytes (limited by the V1 channel of Profibus DP). Data volumes exceeding this maximum size can, for example, be achieved by fragmented into multiple field bus objects of 240 bytes each.

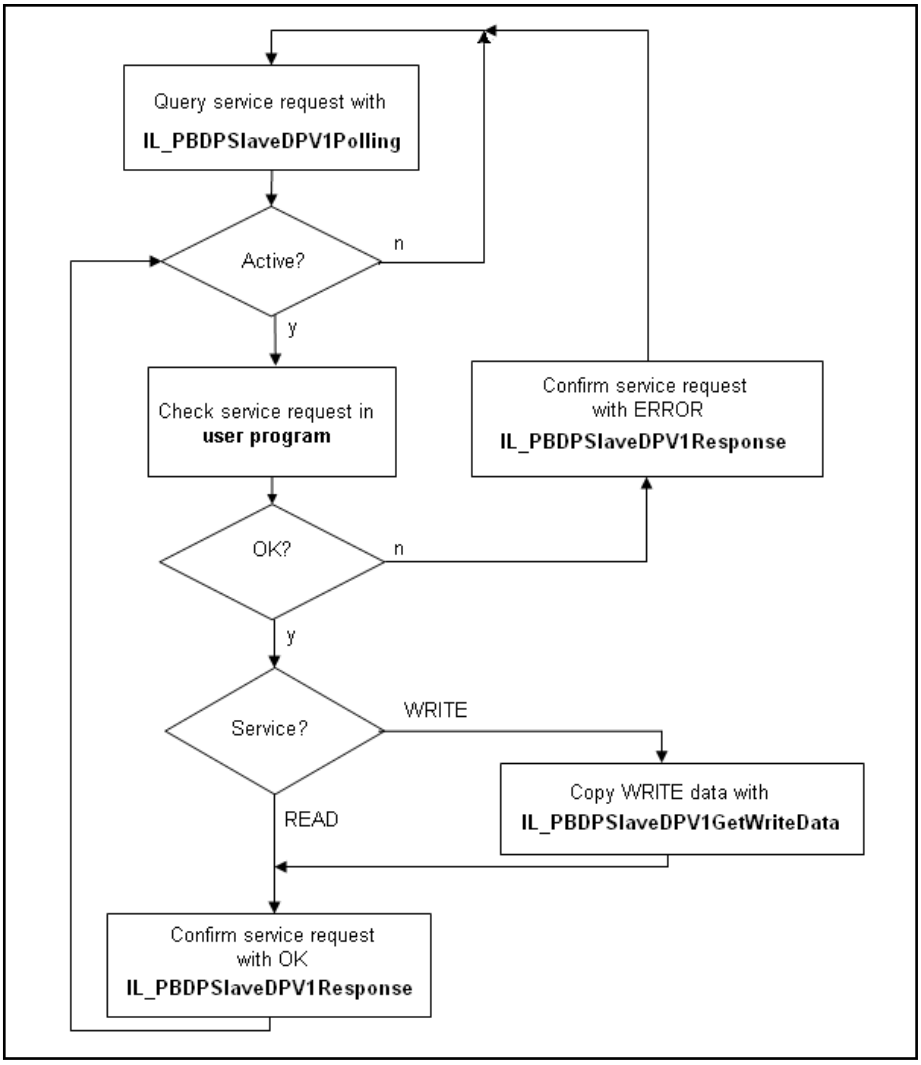


Fig. 4-1: Cooperation of the function blocks of the RIL_ProfibusDPSlave library

The library contains the following components:

- FB IL_PBDPSlaveDPV1Polling, page 59, the polling FB is used to query the activity of a DPV1 service request. The call is made cyclically.
- FB IL_PBDPSlaveDPV1Response, page 61, the response FB is used to respond to an active DPV1 service request. The call is made in relation to the result of the polling FB.
- FB IL_PBDPSlaveDPV1GetWriteData, page 64, GetWriteData is used to copy the data of a DPV1 WRITE service request to the application object.

- FB IL_FBUS_SLAVE_RESULT, page 67 encodes an identifier of the result of the data transfer in the V1 channel.
- FB IL_FBUS_SLAVE_SERVICE, page 67 represents a service identifier of the DPV1 channel.

Selecting the slave

The enumeration type IL_BUSSLAVE of the RIL_FieldbusTypes.library is used to select the respective Profibus DP slave.

Also refer to: FB IL_BUSSLAVE, page 66

4.2 IL_PBDPSlaveDPV1Polling

Brief description

The DPV1 polling FB is used to poll the activity of a DPV1 service request. The FB indicates that a service request is active by output parameters Service, Slot, Index, ServiceID, and DataLength.

Parameter Service defines the type of access (read or write) or is idle while no service request is active.

Parameters Slot and Index define the data object to be accessed.

ServiceID is an internal Ident parameter which must be transferred unchanged when FBs Response and GetWriteData are transferred.

DataLength specifies the access length, i.e., the number of data to be written or the maximum number of data to be read.

The user program is intended to check an active service request for permissibility and to respond to it by calling FB IL_PBDPSlaveDPV1Response (see below).

Target system	Library/folder
MLC	RIL_ProfibusDPSlave
MTX	RIL_ProfibusDPSlave

Tab. 4-1: Reference table

Interface description

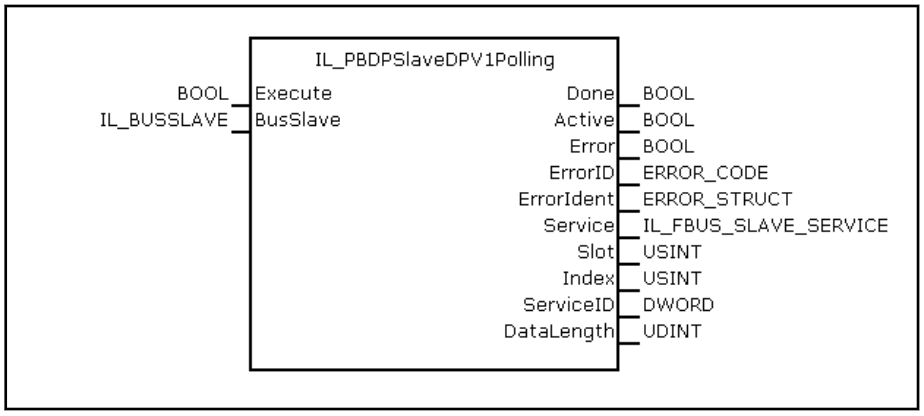


Fig. 4-2: FB IL_PBDPSlaveDPV1Polling interface

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled
	BusSlave	IL_BUSSLAVE	Instance of the slave in the order of the configuration tree. If only one Profibus DP slave is available, the value of this slave is always IL_BUSSLAVE_0. FB IL_BUSSLAVE, page 66
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_ CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_ STRUCT	Detailed diagnostics

I/O type	Name	Data type	Description
	Service	IL_FBUS_SLAVE_SERVICE	Service identifier (read, write or idle) FB IL_FBUS_SLAVE_SERVICE, page 67
	Slot	USINT	Address parameter Slot
	Index	USINT	Address parameter Index
	ServiceID	DWORD	Ident parameter ServiceID
	DataLength	UDINT	Data length (possible data lengths are ≤ 240 bytes)

Tab. 4-2: FB IL_PBDPSlaveDPV1Polling I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	TRUE state of Enable

Tab. 4-3: Min/max values and default values of the IL_PBDPSlaveDPV1Polling inputs

Functional description

- When there is a rising edge at "Execute", IL_PBDPSlaveDPV1Polling is called with the call being effective and the parameter BusSlave is applied.
- If Execute = FALSE, the call does not have any effect. In this case, all output parameters are set to '0'.
- Initially, the result of the call is shown in output parameters Done, Active and Error. If Active = FALSE, the call is completed and the other parameters have to be evaluated.
- If Done = TRUE and Error = FALSE, no error occurred during the call.
- If Done = FALSE and Error = TRUE, errors occurred during the call. In this case, the parameters ErrorID and ErrorIdent describe the error cause.
- If no error occurred during the call, the effective output parameters Service, Slot, Index, ServiceID and DataLength are valid and can be evaluated by the calling instance.
- If errors occurred during the call, the effective output parameters are invalid.
- If output parameters are valid, Service shows the request type (READ, WRITE or IDLE). If the type is READ or WRTE, Slot and Index define the object to be accessed and DataLength defines the access length.
- If Service = IDLE, there is no active service request. In this case, a treatment by the user program is not required.

Troubleshooting

If Error = TRUE, errors occurred during the call.

The errors of the function block can be found in error table PB_DP_TABLE (ERROR_TABLE = 16#0130).

In this case, ErrorID and ErrorIdent (Additional1 and Additional2) define the error cause.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#1001	BusSlave instance	Invalid BusSlave instance

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#1011	0	Repeated polling of an active service
SYSTEM_ERROR	16#1021	0	Slave interface not available
SYSTEM_ERROR	16#1022	Interface error	Slave interface error

Tab. 4-4: FB IL_PBDPSlaveDPV1Polling error codes

4.3 IL_PBDPSlaveDPV1Response

Brief description

The DPV1 Response FB is used to respond to an active DPV1 service request that was detected with IL_PBDPSlaveDPV1Polling beforehand (see above). Depending on whether or not the request is permissible, the FB is called with different parameters ValueAdr, SizeOfValue and Result.

If the "READ" service request is permissible, a pointer to the data object is transferred to the FB in parameter ValueAdr and the corresponding data length in parameter SizeOfValue. The data is copied by the Response FB.

When the "WRITE" service is active, the data is copied by FB IIL_PBDPSlaveDPV1GetWriteData (see below) before the Response FB is called. In this case, parameter ValueAdr is of no relevance.

The response to an impermissible request is given with ValueAdr = 0, SizeOfValue = 0 and a corresponding error identifier in Result.

Target system	Library/folder
MLC	RIL_ProfibusDPSlave
MTX	RIL_ProfibusDPSlave

Tab. 4-5: Reference table

Interface description

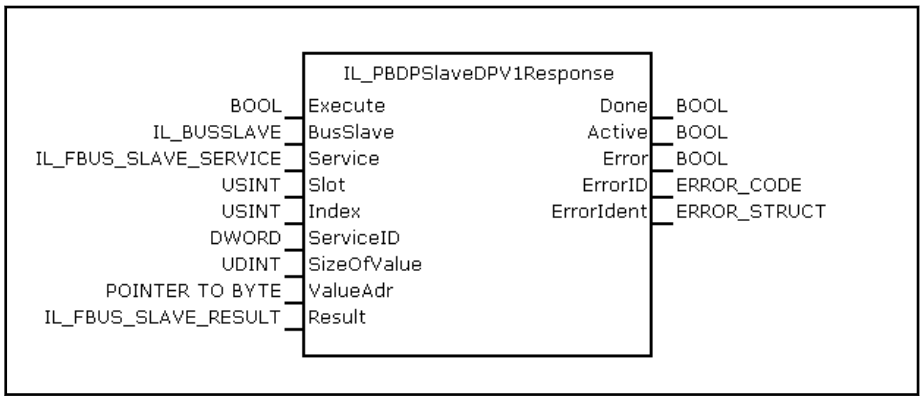


Fig. 4-3: IFB L_PBDPSlaveDPV1Response interface

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled
	BusSlave	IL_BUSSLAVE	Instance of the slave in the order of the configuration tree. If only one Profibus DP slave is available, the value of this slave is always IL_BUSSLAVE_0. FB IL_BUSSLAVE, page 66

I/O type	Name	Data type	Description
	Service	IL_FBUS_SLAVE_SERVICE	Identifier of the service to be completed. This value is compared with the active service. FB IL_FBUS_SLAVE_SERVICE, page 67
	Slot	USINT	Address parameter Slot of the service to be completed. This value is compared with the active Slot.
	Index	USINT	Address parameter Index of the service to be completed. This value is compared with the active Index.
	ServiceID	DWORD	Ident parameter ServiceID of the service for which the data is to be copied (parameter of IL_PBDPSlaveDPV1Polling). This value is compared with the active ServiceID.
	SizeOfValue	UINT	Maximum size of the ValueAdr data range
	ValueAdr	POINTER TO BYTE	Pointer to data
	Result	IL_FBUS_SLAVE_RESULT	Result of the access (is returned to the requesting Profibus master via the V1 response) FB IL_FBUS_SLAVE_RESULT, page 67
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics

Tab. 4-6: FB IL_PBDPSlaveDPV1Response I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Execute
Service	IL_FBUS_SLAVE_SERVICE	IL_SLAVE_SERVICE_IDLE	IL_SLAVE_SERVICE_WRITE	IL_SLAVE_SERVICE_IDLE	Rising edge at Execute
Slot	USINT	0	255	0	Rising edge at Execute
Index	USINT	0	255	0	Rising edge at Execute
ServiceID	DWORD	0	16#FFFFFFFF	0	Rising edge at Execute
SizeOfValue	UDINT	0 (1)	240	0	Rising edge at Execute
ValueAdr	POINTER TO BYTE		N. def.	0	Rising edge at Execute
Result	IL_FBUS_SLAVE_RESULT	IL_SLAVE_RESULT_OK	N. def.	IL_SLAVE_RESULT_OK	Rising edge at Execute

Tab. 4-7: Min/max values and default values of the IL_PBDPSlaveDPV1Response inputs

Functional description

- When there is a rising edge at "Execute", IL_PBDPSlaveDPV1Response is called with the call being effective and the input parameters are applied.

- If Execute = FALSE, the call does not have any effect. In this case, all output parameters are set to '0'.
- Input parameters Service, Slot and Index define the data object having been accessed. It is imperative that these input parameters are identical with the corresponding output parameters of IL_PBDPSlaveDPV1Polling.
- ServiceID is the Ident parameter of the active service. It is imperative that this parameter is identical with the corresponding output parameter of IL_PBDPSlaveDPV1Polling
- In case of a WRITE service, SizeOfValue is identical with DataLength of IL_PBDPSlaveDPV1Polling, but may also be smaller in case of a READ service.
- ValueAdr is a pointer to the data object addressed and is '0' in case of an invalid service request.
- Result defines an error event during access to the requested data object.
- Initially, the result of the call is shown in output parameters Done, Active and Error.

  If Active = FALSE, the call is completed and the other parameters have to be evaluated.
- If Done = TRUE and Error = FALSE, no error occurred during the call.
- If Done = FALSE and Error = TRUE, errors occurred during the call.

  In this case, the parameters ErrorID and ErrorIdent describe the error cause.

Troubleshooting

If Error = TRUE, errors occurred during the call.

The errors of the function block can be found in error table PB_DP_TABLE (ERROR_TABLE = 16#0130).

In this case, ErrorID and ErrorIdent (Additional1 and Additional2) define the error cause.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#2001	BusSlave instance	Invalid BusSlave instance
INPUT_RANGE_ERROR	16#2002	Service	Invalid service
INPUT_RANGE_ERROR	16#2003	Slot	Invalid Slot
INPUT_RANGE_ERROR	16#2004	Index	Invalid index
INPUT_RANGE_ERROR	16#2005	ServiceID	Invalid ServiceID
INPUT_RANGE_ERROR	16#2006	SizeOfValue	Invalid SizeOfValue
INPUT_RANGE_ERROR	16#2007	ValueAdr	Invalid ValueAdr
INPUT_RANGE_ERROR	16#2008	Result	Invalid Result
STATE_MACHINE_ERROR	16#2011	0	No service active
COMMUNICATION_ERROR	16#2012	Timeout [ms]	Service not processed in time

ErrorID	Additional1	Additional2	Description
SYSTEM_ERROR	16#2021	0	Slave interface not available
SYSTEM_ERROR	16#2022	Interface error	Slave interface error

Tab. 4-8: FB IL_PBDPSlaveDPV1Response error codes

4.4 IL_PBDPSlaveDPV1GetWriteData

Brief description

The DPV1 GetWriteData FB is used to copy the user data to the appropriate target object in case of an active and permissible DPV1 WRITE service request that was detected with IL_PBDPSlaveDPV1Polling before.

To achieve this, a pointer to the data object is transferred to the FB in parameter ValueAdr and the corresponding data length in parameter SizeOfValue.

Target system	Library/folder
MLC	RIL_ProfibusDPSlave
MTX	RIL_ProfibusDPSlave

Tab. 4-9: Reference table

Interface description

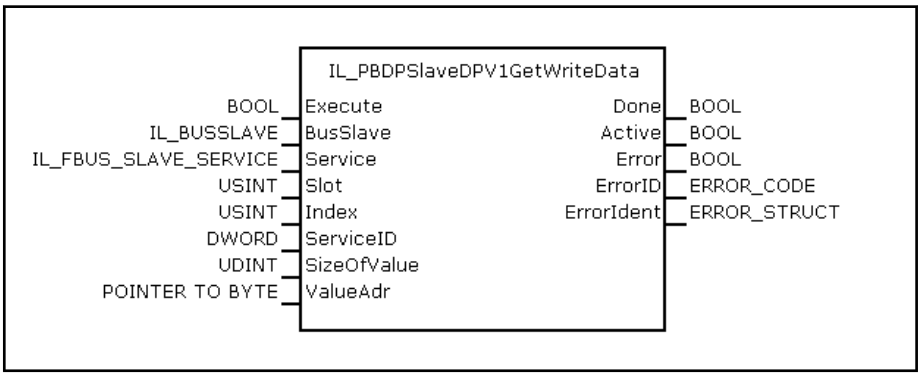


Fig. 4-4: FB IL_PBDPSlaveDPV1GetWriteData interface

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the slave in the order of the configuration tree. If only one Profibus DP slave is available, the value of this slave is always IL_BUSSLAVE_0. FB IL_BUSSLAVE, page 66
	Service	IL_FBUS_SLAVE_SERVICE	Identifier of the service for which the data is to be copied. This value is compared with the active service. FB IL_FBUS_SLAVE_SERVICE, page 67
	Slot	USINT	Address parameter Slot of the service for which the data is to be copied. This value is compared with the active Slot.
	Index	USINT	Address parameter Index of the service for which the data is to be copied. This value is compared with the active Index.
	ServiceID	DWORD	Ident parameter ServiceID of the service for which the data is to be copied (parameter of IL_PBDPSlaveDPV1Polling). This value is compared with the active ServiceID.
	SizeOfValue	UINT	Maximum size of the ValueAdr data range

I/O type	Name	Data type	Description
	ValueAdr	POINTER TO BYTE	Pointer to data
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_ CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_ STRUCT	Detailed diagnostics

Tab. 4-10: FB IL_PBDPSlaveDPV1GetWriteData I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Execute
Service	IL_FBUS_SLAVE_SERVICE	IL_SLAVE_SERV-ICE_IDLE	IL_SLAVE_SERV-ICE_WRITE	IL_SLAVE_SERV-ICE_IDLE	Rising edge at Execute
Slot	USINT	0	255	0	Rising edge at Execute
Index	USINT	0	255	0	Rising edge at Execute
ServiceID	DWORD	0	16#FFFFFFFF		Rising edge at Execute
SizeOfValue	UDINT	0 (1)	240	0	Rising edge at Execute
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Rising edge at Execute

Tab. 4-11: Min/max values and default values of the IL_PBDPSlaveDPV1Get-WriteData inputs

Functional description

- When there is a rising edge at "Execute", IL_PBDPSlaveDPV1GetWrite-Data is called with the call being effective and the input parameters are applied.
- If Execute = FALSE, the call does not have any effect. In this case, all output parameters are set to '0'.
- If Execute = TRUE (constantly), the output parameters remain as they are.
- Input parameters Service, Slot and Index define the data object having been accessed. It is imperative that these input parameters are identical with the corresponding output parameters of IL_PBDPSlaveDPV1Poll-ing.
- ServiceID is the Ident parameter of the active service. It is imperative that this parameter is identical with the corresponding output parameter of IL_PBDPSlaveDPV1Polling.
- In case of a WRITE service, SizeOfValue is identical with DataLength of IL_PBDPSlaveDPV1Polling, but may also be smaller in case of a READ service.
- ValueAdr is a pointer to the target object.
- Initially, the result of the call is shown in output parameters Done, Active and Error. If Active = FALSE, the call is completed and the other param-eters have to be evaluated.

- If Done = TRUE and Error = FALSE, no error occurred during the call.
- If Done = FALSE and Error = TRUE, errors occurred during the call. In this case, the parameters ErrorID and ErrorIdent describe the error cause.

Troubleshooting If Error = TRUE, errors occurred during the call.

The errors of the function block can be found in error table PB_DP_TABLE (ERROR_TABLE = 16#0130).

In this case, ErrorID and ErrorIdent (Additional1 and Additional2) define the error cause.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#3001	BusSlave instance	Invalid BusSlave instance
INPUT_RANGE_ERROR	16#3002	Service	Invalid service
INPUT_RANGE_ERROR	16#3003	Slot	Invalid Slot
INPUT_RANGE_ERROR	16#3004	Index	Invalid index
INPUT_RANGE_ERROR	16#3005	ServiceID	Invalid ServiceID
INPUT_RANGE_ERROR	16#3006	SizeOfValue	Invalid SizeOfValue
INPUT_RANGE_ERROR	16#3007	ValueAdr	Invalid ValueAdr
STATE_MACHINE_ERROR	16#3011	0	No service active
COMMUNICATION_ERROR	16#3012	Timeout [ms]	Service not processed in time
SYSTEM_ERROR	16#3021	0	Slave interface not available
SYSTEM_ERROR	16#3022	Interface error	Slave interface error

Tab. 4-12: FB IL_PBDPSlaveDPV1GetWriteData error codes

4.5 IL_BUSSLAVE

Brief description The IL_BUSSLAVE enumeration type serves to select the particular Profibus DP slave. In addition to the onboard Profibus interface, other slaves can also be available in a control system via function modules.

The Profibus DP slaves are distinguished based on their ascending order in the configuration.

Indexes are assigned to the devices in the following order:

- Onboard slave
- Function module 1
- Function module 2
- Function module 3
- Function module 4

If, for example, the onboard slave is not a Profibus slave while a Profibus slave is configured on the FM1, IL_BUSSLAVE_0 is the value assigned to the slave on the FM1.

Element	Value	Description
IL_BUSSLAVE_0	0	First device
IL_BUSSLAVE_1	1	Second device
IL_BUSSLAVE_2	2	Third device
IL_BUSSLAVE_3	3	Fourth device
IL_BUSSLAVE_4	4	Fifth device

Tab. 4-13: Elements of the IL_BUSSLAVE enumeration type

Examples of addressing the various instances of the Profibus slaves on a control:

Configured:	Addressed with:
Onboard: Profibus DP slave	IL_BUSSLAVE_0
FM1: Profibus DP master	IL_BUSMASTER_0
FM2: Profibus DP slave	IL_BUSSLAVE_1
FM3: Profibus DP slave	IL_BUSSLAVE_2

Tab. 4-14: Example 1 of IL_BUSSLAVE enumeration type

4.6 IL_FBUS_SLAVE_SERVICE

Brief description The IL_FBUS_SLAVE_SERVICE enumeration type serves to represent a service identifier of the DPV1 channel. This indicates the request state currently existing in the V1 channel.

The bits are in the "RIL_FIELDBUS_TYPES" library as defines.

Element	Value	Description
IL_SLAVE_SERVICE_IDLE	0	No active request
IL_SLAVE_SERVICE_READ	1	A read request is present
IL_SLAVE_SERVICE_WRITE	2	A write request is present

Tab. 4-15: Elements of data type IL_FBUS_SLAVE_SERVICE

4.7 IL_FBUS_SLAVE_RESULT

Brief description The IL_FBUS_SLAVE_RESULT enumeration type serves to encode an identifier of the result of the data transfer in the V1 channel. This allows the user to specify whether data transfer was faultless and which error code is returned to the requesting Profibus master via the V1 channel.

The bits are in the "RIL_FIELDBUS_TYPES" library as defines.

Element	Value	Description
IL_SLAVE_RESULT_OK	16#00	No error during access
IL_SLAVE_RESULT_INVALID_SLOT	16#B2	Faulty Slot
IL_SLAVE_RESULT_INVALID_INDEX	16#B0	Faulty Index
IL_SLAVE_RESULT_ACCESS_DENIED	16#B6	Access not allowed
IL_SLAVE_RESULT_READ_ONLY	16#BA	Write access not allowed

Element	Value	Description
IL_SLAVE_RESULT_READ_ERROR	16#A0	Error during reading (with respect to the PLC application)
IL_SLAVE_RESULT_WRITE_ERROR	16#A1	Error during writing (with respect to the PLC application)
IL_SLAVE_RESULT_WRITE_LENGTH_ERROR	16#B1	Write access with excessive data length (the number of data to be written exceeds the number allowed for the data object)
IL_SLAVE_RESULT_READ_LENGTH_ERROR	16#BB	Read access with insufficient data length (the number of data to be read is less than the number required for the data object)
IL_SLAVE_RESULT_RESOURCE_BUSY	16#C2	Data access not possible at present
IL_SLAVE_RESULT_USER_ERROR_0	16#F0	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_1	16#F1	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_2	16#F2	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_3	16#F3	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_4	16#F4	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_5	16#F5	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_6	16#F6	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_7	16#F7	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_8	16#F8	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_9	16#F9	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_10	16#FA	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_11	16#FB	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_12	16#FC	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_13	16#FD	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_14	16#FE	Meaning is user-defined
IL_SLAVE_RESULT_USER_ERROR_15	16#FF	Meaning is user-defined

Tab. 4-16: Elements of data type IL_FBUS_SLAVE_RESULT

5 RIL_ProfinetIO.library

5.1 Introduction and overview

This library describes the IEC program options allowing access to the netX Profinet I/O controller.

The implementation of the library was based on RIL_UTILITIES.

The IL_EnableDoneBase and IL_ExecuteDoneBase methods of the basic function blocks (basic classes) were used and extended or overwritten.



This note applies to the following Profinet function blocks:

- Acyclic communication
- Backup & Restore
- Device Replacement
- IO-Link,
- IM-Data,
- IL_PNIORemoteDeviceStateDetails
- IH_PNIOAck_PSDx_Inline
- IH_PNIOAck_PSDx_S20

While programming, note that only an acyclic service can be executed (DCP services or the Read-Implicit service). A new acyclic request can only be started after the previously started request has been completed. Thus, start a new function block with a positive edge at “Execute” only after the processing and all other function blocks used have been completed. The processing is completed when either the “Done” output or the “Error” output is set.

Areas of application and function blocks of RIL_ProfinetIO

Area of application (folder in library)	
Function blocks	**Description**
Acyclic services (Acyclic Communication)	
IL_PNIOWriteRecord, page 79	Acyclic writing
IL_PNIOReadRecord, page 71	Acyclic reading
IL_PNIOServiceRecord, page 74	Used for the Profinet service interface access, which is for example required for the IO-Link integration or for the PROFIenergy
Backing up and restoring (Backup_Restore)	
IL_PNIODeviceBackup, page 82	Used to backup the configuration and the parameters of a Profinet I/O device to a file via acyclic reading
IL_PNIODeviceRestore, page 86	Used to restore parameters of a Profinet I/O device via acyclic writing from the backup file
Supporting the device replacement	
IL_PNIOAssignName, page 89	Assigns a device name (ident string) to a Profinet I/O device
IL_PNIOCheckTopology, page 91	Checks the saved topology of the Profinet device replacement function using the currently available Profinet topology. This is executed by reading the port information from the Profinet I/O device

IL_PNIODeviceReplacement, page 96	Provides a diagnostics via the status of the automatic device replacement function. The device replacement function can also be disabled
IL_PNIODeviceResetToDefault, page 98	Resets a Profinet I/O device to the factory default
IL_PNIODeviceSetIP, page 100	Sets the IP settings of a Profinet I/O device
IL_PNIOGetLiveList, page 102	Determining the list of currently accessible Profinet I/O devices
IL_PNIOGetLogMsg, page 104	Gets a logger text
IL_PNIOGetPortData, page 105	Provides port data ("neighboring information") from a Profinet I/O device
IL_PNIOStoreTopology, page 108	Generates the network topology information and saves it to an XML file
IL_PNIORestoreTopology, page 109	Tries to restore the saved topology of the PNIO controller (e.g. use case "cross-exchange" of PN devices).
Diagnostics	
IL_PNIOControllerState, page 114	Determining the simple status
IL_PNIOControllerStateDetails, page 116	Determining the status of the Profinet stack
IL_PNIODiagnosis, page 118	Creates a comprehensive diagnostics of a Profinet I/O controller and of its connected devices
IL_PNIOGetConfigDeviceNameList, page 122	Determining the list of configured Profinet I/O devices
IL_PNIOGetControllerList, page 124	Provides the number of configured Profinet I/O controllers
IL_PNIOGetDiagDeviceNameList, page 125	Determining the list of diagnostic-reporting Profinet I/O devices
IL_PNIORemoteDeviceState, page 126	Determining the simple status of any Profinet I/O device
IL_PNIORemoteDeviceStateDetails, page 128	Determining the status of any Profinet I/O device
I&M (Identification and Maintenance) (IM Data)	
IL_PNIOReadWriteIMData, page 132	Can read or write I&M data sets (identification & maintenance) of the Profinet I/O device with different read and write modes
Communication interface (IO-Link)	
IL_PNIOIoLinkBackupDevice, page 175	Used to backup the configuration and parameters of the IOL device to a file via Profinet I/O IO-Link gateways
IL_PNIOIoLinkCall, page 139	Is responsible for read and write services of the IOL data objects via Profinet I/O IO-Link gateways
IL_PNIOIoLinkGetDeviceAccessLock, page 145	Reads the status of the IO-Link device parameter “Device access locks” of the IO-Link device connected to the Profinet coupler
IL_PNIOIoLinkIdent, page 150	Is used to read basic information to identify an IO-Link device connected at a specified Profinet IO-Link port
IL_PNIOIoLinkIdentDetails, page 155	Is used to read identification parameters of an IO-Link device connected to a specified Profinet IO-Link port
IL_PNIOIoLinkRemoteDeviceState, page 160	Reads the current diagnostic status of the IO-Link device connected to a specified IO-Link port on Profinet
IL_PNIOIoLinkRestoreDevice, page 180	Used to restore the parameters of an IOL device from the backup file via the Profinet I/O IO-Link gateway

IL_PNIOIoLinkSetDeviceAccessLock, page 165	Allows to control the Profinet I/O device behavior by overwriting the parameters "Device access locks" of the IO-Link device connected to the Profinet coupler
IL_PNIOIoLinkSystemCommand, page 170	Is used to write a system command to an IO-Link device connected to a specified Profinet IO-Link port

Tab. 5-1: RIL_ProfinetIO library

Selecting the Profinet I/O controller

Addressing: The Profinet I/O controllers are distinguished based on their ascending order in the configuration:

Counting starts with "0" at the first Profinet I/O controller.

If only one Profinet I/O controller is configured, the controller instance is always "0" (also refer to IL_BUSMASTER, page 184).

Addressing the Profinet I/O devices

Device name: Profinet uses the "IdentStr" device name to address the devices.

Enter the device name into the configuration, see "Station name".

PROFINETIO_TABLE 16#200

The PROFINETIO_TABLE 16#200, page 190, contains error numbers output by the function blocks of the library.

5.2 Acyclic services (Acyclic Communication)

5.2.1 IL_PNIOReadRecord

Brief description

The function block **IL_PNIOReadRecord** reads acyclic data from Profinet I/O devices.

The Profinet I/O devices are addressed via their Profinet names in "IdentStr".

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/AcyclicCommunication
MTX	RIL_ProfinetIO/POUs/AcyclicCommunication

Tab. 5-2: Reference table

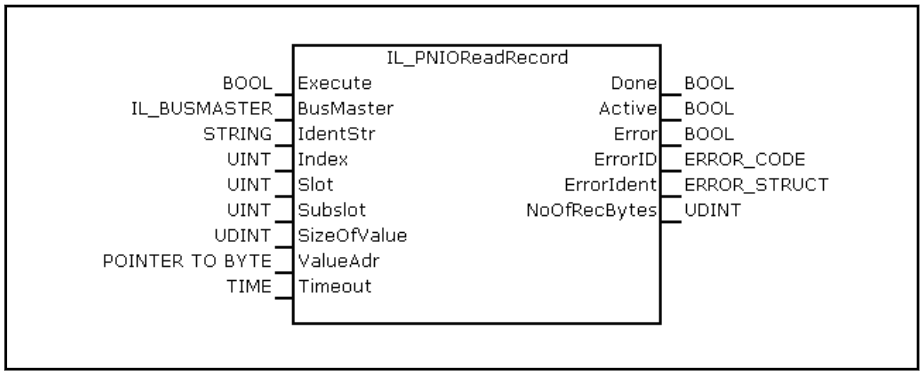


Fig. 5-1: Interface description of the IL_PNIOReadRecord function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the field bus master in the order of the configuration tree Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	IdentStr	STRING	Identification string of the field bus device
	Index	UINT	Index of the addresses data set. Indices 16#8000 to 16#FFFF are reserved for Profinet. Indices from 0 to 16#7FFF are user- or application-specific
	Slot	UINT	Addressing the module slot. A slot addresses a logical or a physical module of a Profinet I/O device (e.g. an I/O module)
	Subslot	UINT	Addressing the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	SizeOfValue	UDINT	Size of the data range addressed with "ValueAdr"
	ValueAdr	POINTER TO BYTE	Initial address of the data range
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NoOf RecBytes	UDINT	Number of bytes received in the data range

Tab. 5-3: I/O interface of the IL_PNIOReadRecord function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
IdentStr	STRING			’’	Rising edge at "Execute"
Index	UINT	0	16#ffff	0	Rising edge at "Execute"
Slot	UINT	1	16#7fff	0	Rising edge at "Execute"
Subslot	UINT	1	16#8fff	0	Rising edge at "Execute"
SizeOfValue	UDINT	0	N. def.	0	Rising edge at "Execute"

Name	Type	Min. value	Max. value	Default value	Effective
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at "Execute"

Tab. 5-4: Min./max. and default values of the IL_PNIOReadRecord inputs

Functional description

The Profinet I/O device is addressed via the device name in "IdentStr". It has to be part of the configuration of the Profinet I/O controller. The data set on the Profinet I/O device is addressed via the “Slot” and “SubSlot” combination as well as via the “Index”.

"Slot" and "Subslot" address the various logical or physical modules of a Profinet I/O device. The range from 16#8000 to 16#ffff is assigned by IEC 61158-6-10 V23. The range from 16#8000 to 16#ffff is specified by IEC 61158-6-10 V23. The value range from 0 to 16#7fff can be specified by the manufacturer of the Profinet I/O device. For more information, refer to the corresponding documentation of the Profinet I/o device.

The I&M0 data of the Profinet I/O device can for example be triggered with the index 16#AFF0 as well as with "Slot 0" and "Subslot 1".

The data is requested in the maximum length allowed ("SizeOfValue") by the Profinet I/O device. The number of valid data bytes actually provided by the device is returned in "NoOfRecBytes".

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	0	"ValueAdr" is not set
INPUT_RANGE_ERROR	16#0001	1	"SizeOfValue" is not set
INPUT_RANGE_ERROR	16#0001	2	"IdentStr" is invalid
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller is not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	16#0001 16#0002	Error in the state machine
COMMUNICATION_ERROR	16#0003	**Profinet I/O controller diagnostics** • Refer to chapter 5.9.2 "General Error Codes " on page 191 • Refer to chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	**Profinet I/O status** • Refer to chapter 5.9.4 "Special error codes, Profinet IO status " on page 193	

Tab. 5-5: Error codes of the IL_PNIOReadRecord function block

5.2.2 IL_PNIOServiceRecord

Brief description The function block **IL_PNIOServiceRecord** is used for the Profinet service interface access, which is, for example, required for the IO-Link integration or for the PROFIenergy.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/IO-Link
MTX	RIL_ProfinetIO/POUs/IO-Link

Tab. 5-6: Reference table

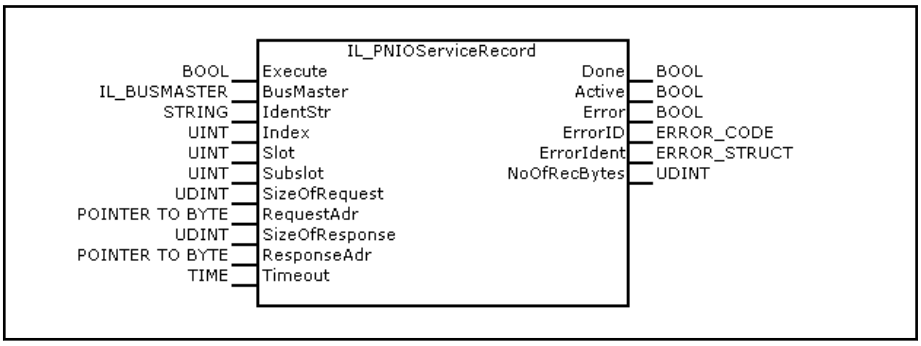


Fig. 5-2: Interface description of the IL_PNIOServiceRecord function block

I/O type	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	IdentStr	STRING	Identification string of the Profinet I/O device
	Index	UINT	Index of the addresses data set. Indices 16#8000 to 16#FFFF are reserved for Profinet. Indices from 0 to 16#7FFF are user- or application-specific
	Slot	UINT	Addressing the module slot. The slot addresses a logical or a physical module (e.g. an I/O module)
	SubSlot	UINT	Addressing the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	SizeOfRequest	UDINT	Length of request data. (<= SizeOf(RequestAdr))
	RequestAdr	POINTER TO BYTE	Initial address of the requested data range. Use Adr( ) operator
	SizeOfResponse	UDINT	Size of the response address array. Use Adr SizeOf( ) operator
	ResponseAdr	POINTER TO BYTE	Initial address of the response data range. Use Adr( ) operator
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	NoOfRecBytes	UDINT	Number of bytes received as response

Tab. 5-7: I/O interface of the IL_PNIOServiceRecord function block

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
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster 12 16	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"

Name	Type	Min. value	Max. value	Default value	Effective
IdentStr	STRING	---		---	Rising edge at "Execute"
Index	UINT	0	16#ffff	0	Rising edge at "Execute"
Slot	UINT	0	16#7fff	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8fff	1	Rising edge at "Execute"
SizeOfRequest	UDINT	0	----	0	Rising edge at "Execute"
RequestAdr	POINTER TO BYTE	0	----	0	Rising edge at "Execute"
SizeOfResponse	UDINT	0	----	0	Rising edge at "Execute"
ResponseAdr	POINTER TO BYTE	0	----	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	----	T#2s	Rising edge at "Execute"

Tab. 5-8: Min./max. and default values of the IL_PNIOServiceRecord inputs

Functional description

The function block **IL_PNIOServiceRecord** is used for the Profinet service interface access, which is, for example, required for the IO-Link integration or for the PROFIenergy. The service interface can be accessed via the function blocks IL_PNIOWriteRecord (write service request) and IL_PNIOReadRecord (read service request).

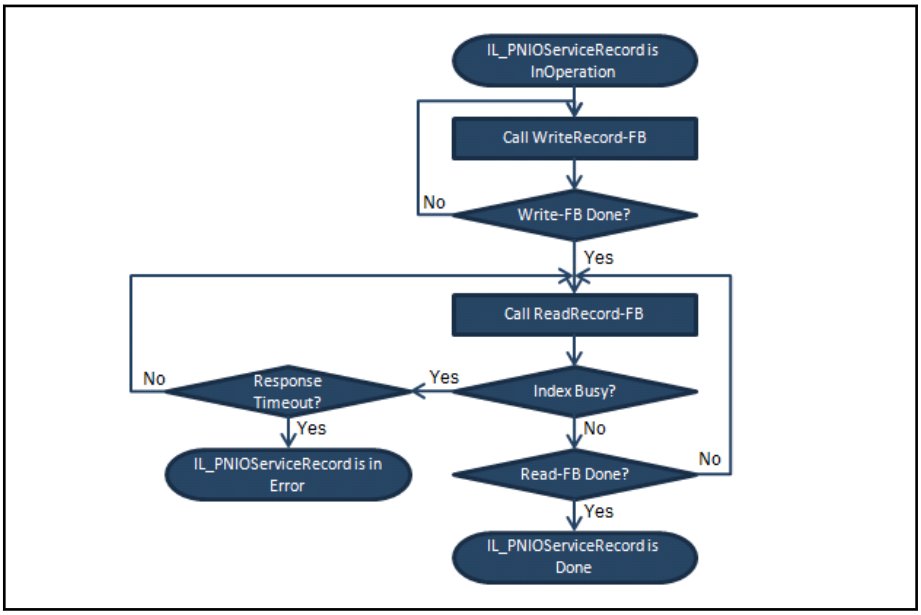


*Fig. 5-3: Sequence diagram of the function block **IL_PNIOServiceRecord** of the RIL_ProfinetIO.library*

The Profinet I/O device is addressed via the Profinet device name in "IdentStr". It has to be part of the configuration of the Profinet I/O controller. The data set on the Profinet I/O device is addressed via the "Slot" and "SubSlot" combination as well as via the "Index".

"Slot" and "SubSlot" address the various logical or physical modules of a Profinet I/O device. The range from 16#8000 to 16#ffff is assigned by IEC 61158-6-10 V23. The range from 16#8000 to 16#ffff is specified by IEC 61158-6-10 V23. The value range from 0 to 16#7fff can be specified by the manufacturer of the Profinet I/O device. For more information, refer to the corresponding documentation of the Profinet I/o device. The specified data range "RequestAdr" is transferred to the Profinet I/O device with its complete size "SizeOfValue". The reaction of the Profinet I/O device to exceeded ranges is manufacturer-specific.

The data is requested in the maximum length allowed ("SizeOfResponse") by the Profinet I/O device. The number of valid data bytes actually provided by the device is returned in "NoOfRecBytes".

Example program

The function block **IL_PNIOServiceRecord** is used for the Profinet service interface access.

Example: **IL_PNIOServiceRecord** reads the manufacturer name of the Profinet I/O device connected to port 2 of the IOL master.

Declaration

```
VAR
    fbIL_PNIOServiceRecord: IL_PNIOServiceRecord;
    bExecute: BOOL;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    arRequestAdr: ARRAY[0..7] OF BYTE;
    udiNoOfRecBytes: UDINT;
    arResponseAdr: array[0..232] of byte;
END_VAR
```

Implementation

```
fbIL_PNIOServiceRecord(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    BusMaster:= IL_BusMaster.IL_BUSMASTER_0,
    IdentStr:='iolballuff',
    Index:= UINT#255,
    Slot:= UINT#1,
    Subslot:= UINT#1,
    SizeOfRequest:= UDINT#8,
    RequestAdr:= ADR(arRequestAdr),
    SizeOfResponse:= SIZEOF(arResponseAdr),
    ResponseAdr:= ADR(arResponseAdr),
    Timeout:= T#2S,
    NoOfRecBytes=> udiNoOfRecBytes);
```

The result of the IL_PNIOServiceRecord is shown in the following figure:

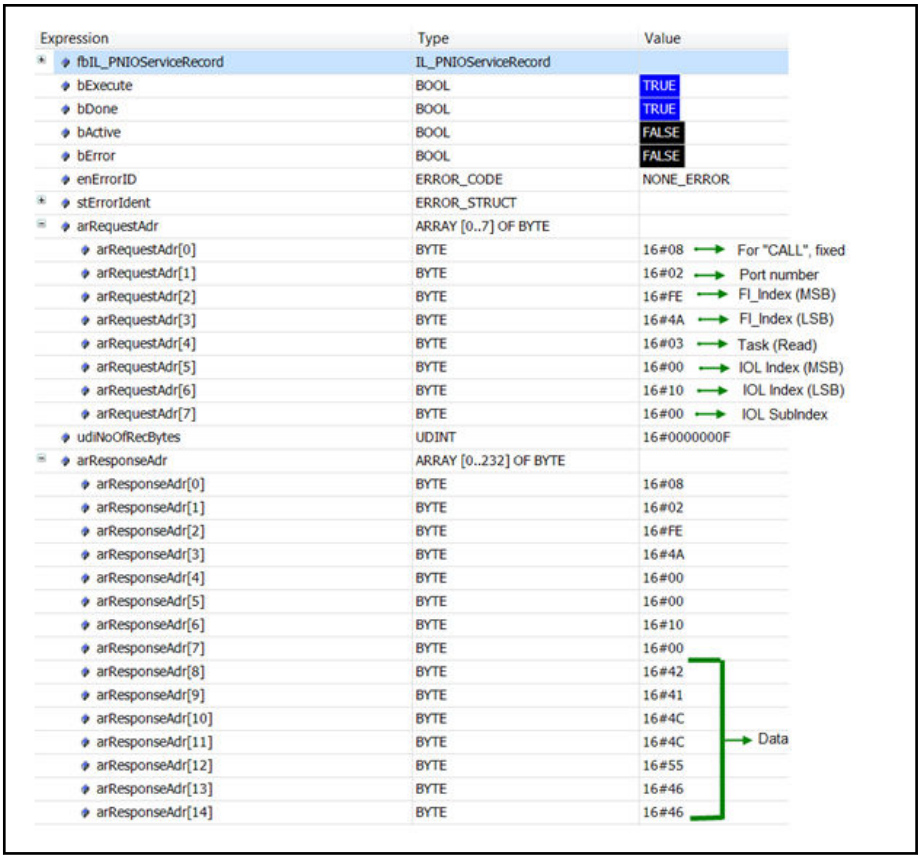


Fig. 5-4: IL_PNIOServiceRecord, example, online

Error handling

The function block **IL_PNIOServiceRecord** uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	16#0000	"ValueAdr" ("RequestAdr" or "ResponseAdr") is not set
INPUT_RANGE_ERROR	16#0001	16#0001	"SizeOfValue" ("SizeOfRequest" or "SizeOfResponse") is not set
INPUT_RANGE_ERROR	16#0001	16#0002	"IdentStr" is invalid
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller is not in configuration table
COMMUNICATION_ERROR	16#0104	16#0000	Timeout
STATE_MACHINE_ERROR	16#0105	Internal use	Error in the state machine

ErrorID	Additional1	Additional2	Note
COMMUNICATION_ERROR	16#0003	**Profinet I/O controller diagnostics** • General error codes, bus master chapter 5.9.2 "General Error Codes " on page 191 • Special error codes, Profinet I/O controller chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	**Profinet I/O status** • Error codes, Profinet I/O status chapter 5.9.4 "Special error codes, Profinet IO status " on page 193	

Tab. 5-9: Error codes of the IL_PNIOServiceRecord

5.2.3 IL_PNIOWriteRecord

Brief description

The function block **IL_PNIOWriteRecord** writes acyclic data to Profinet I/O devices.

The Profinet I/O devices are addressed via their device name. The Profinet I/O devices have to be present in the configuration.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/AcyclicCommunication
MTX	RIL_ProfinetIO/POUs/AcyclicCommunication

Tab. 5-10: Reference table

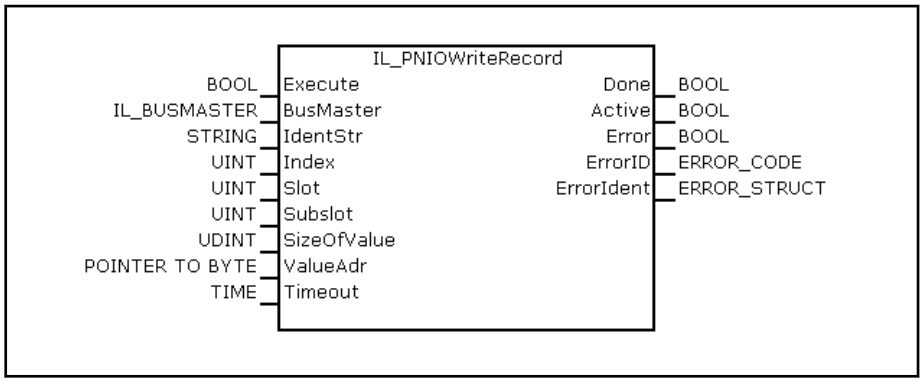


Fig. 5-5: Interface description of the IL_PNIOWriteRecord function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	IdentStr	STRING	Station name of the Profinet I/O device to be accessed
	Index	UINT	Index of the addresses data set. Indices 16#8000 to 16#FFFF are reserved for Profinet. Indices from 0 to 16#7FFF are user- or application-specific
	Slot	UINT	Addressing the module slot. In a modular Profinet I/O device, the physical modules are addressed via "Slot". In a compact Profinet I/O device, "Slot" addresses a logic function or a virtual module
	Subslot	UINT	Addressing the submodule. "Subslot" can be used to address the physical interfaces of the submodules of a module. In general, "Subslot" is the second structure level of a Profinet I/O device
	SizeOfValue	UDINT	Maximum size of the "ValueAdr" data range
	ValueAdr	POINTER TO BYTE	Pointer to the data storage
	Timeout	TIME	Timeout for abortion of the function
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)

Tab. 5-11: I/O interface of the IL_PNIOWriteRecord function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
IdentStr	STRING			’’	Rising edge at "Execute"
Index	UINT	0	16#ffff	0	Rising edge at "Execute"
Slot	UINT	1	16#7fff	0	Rising edge at "Execute"

Name	Type	Min. value	Max. value	Default value	Effective
Subslot	UINT	1	16#8fff	0	Rising edge at "Execute"
NoOfBytes	UDINT	0	N. def.	0	Rising edge at "Execute"
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at "Execute"

Tab. 5-12: Min./max. and default values of the IL_PNIOWriteRecord inputs

Functional description

The Profinet I/O device is addressed via device name. It has to be part of the configuration of the Profinet I/O controller. The data set on the Profinet I/O device is addressed via the "Slot" and "SubSlot" combination as well as via the "Index".

"Slot" and "Subslot" address the various logical or physical modules of a Profinet I/O device. The range from 16#8000 to 16#ffff is assigned by IEC 61158-6-10 V23. The range from 16#8000 to 16#ffff is specified by IEC 61158-6-10 V23. The value range from 0 to 16#7fff can be specified by the manufacturer of the Profinet I/O device. For more information, refer to the corresponding documentation of the Profinet I/o device.

The specified data range "ValueAdr" is transferred to the Profinet I/O device with its complete size "SizeOfValue". The reaction of the Profinet I/O device to exceeded ranges is manufacturer-specific.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	0	"ValueAdr" is not set
INPUT_RANGE_ERROR	16#0001	1	"SizeOfValue" is not set
INPUT_RANGE_ERROR	16#0001	2	"IdentStr" is not set
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller is not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
COMMUNICATION_ERROR	16#0003	**Profinet I/O controller diagnostics (refer to Additional2 code)** • Refer to chapter 5.9.2 "General Error Codes " on page 191 • Refer to chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	**Profinet I/O status** • Refer to chapter 5.9.4 "Special error codes, Profinet IO status " on page 193	

Tab. 5-13: Error codes of the IL_PNIOWriteRecord function blocks

5.3 Backing up and restoring (Backup_Restore)

5.3.1 IL_PNIODeviceBackup

Brief description The function block **IL_PNIODeviceBackup** is used to back up the configuration and the parameters of an I/O device to a file via acyclic reading (chapter 5.2.1 "IL_PNIOReadRecord" on page 71).

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/Backup Restore

Tab. 5-14: Reference table

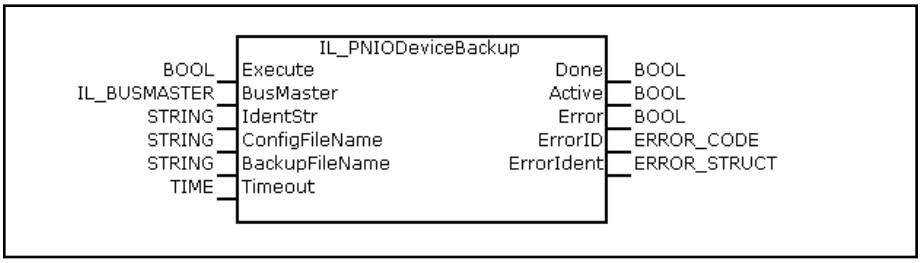


Fig. 5-6: Interface description of the IL_PNIODeviceBackup function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	IdentStr	STRING	Identification string of the Profinet I/O device
	ConfigFileName	STRING	ConfigFile with "Slot", "SubSlot" and "Index" of the parameters to be retrieved from the Profinet I/O device. Name of configuration file without extension (*.pnc)
	BackupFileName	STRING	Name of backup file without extension (*.pnb)
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnostics in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE) "Error handling" on page 85

Tab. 5-15: I/O interface of the IL_PNIODeviceBackup function block

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
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
Timeout	TIME	T#0ms		T#2s	Rising edge at "Execute"

Tab. 5-16: Min./max. and default values of the IL_PNIODeviceBackup inputs

Functional description

The function block **IL_PNIODeviceBackup** is used to backup the configuration and the parameters of a Profinet I/O device to a file after acyclic reading (chapter 5.2.1 "IL_PNIOReadRecord" on page 71) from Profinet I/O device.

The function block retrieves the parameters listed in the configuration file "ConfigFileName" from the Profinet I/O device and saves the data to the file "BackupFileName".

The configuration and backup files are stored on the Compact Flash card of the control (USER-Partition, /ata0b/).

The function block creates a new file upon each execution. If a file already exists, it is overwritten.

The configuration file is stored as text with the extension "*.pnc".

Basic structure of the ConfigFile	Example of the ConfigFile
Slot, subslot, index	16#0000, 16#0001, 16#0040
Slot, subslot, index	16#0000, 16#0001, 16#AFF0
Slot, subslot, index	16#0001, 16#0001, 16#8000
...	...
...	...
...	...
Slot, subslot, index	16#0002, 16#0001, 16#C000

The function block retrieves the parameters from the Profinet I/O device and saves them to the backup file at a rising edge at the "Execute" input.

Basic backup file	Example of the backup file
Slot, SubSlot, Index, DataSize, Data	16#0000, 16#0001, 16#0040, 16#0001, 16#58
Slot, SubSlot, Index, DataSize, Data1, Data2, Data3	16#0000, 16#0001, 16#AFF0,16#0003, 16#20, 16#82, 16#32
...	...
...	...
Slot, SubSlot, Index, DataSize, Data1, Data2	16#0000, 16#0001, 16#AFF0, 16#0002, 16#45, 16#56

Example program

This example describes the creation of the backup of the Profinet I/O device parameters to store them on the Compact Flash card in the IndraControl control.

- Specify the name of the configuration file in "strConfigFileName"
- Save the configuration file on the Compact Flash card (USER partition, /ata0b/) of the control

  Steps to create and store the configuration file

  1. Open notepad and enter backup parameter information.

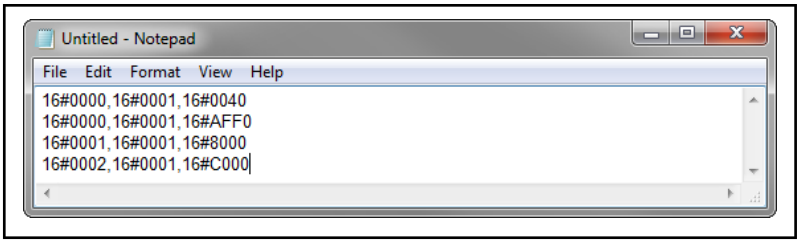


Fig. 5-7: Configuration file

2. Save configuration file as "ConfigFile.enc".
3. Right-click on the control in the Project Explorer and navigate to **Device data ▸ Explorer**.

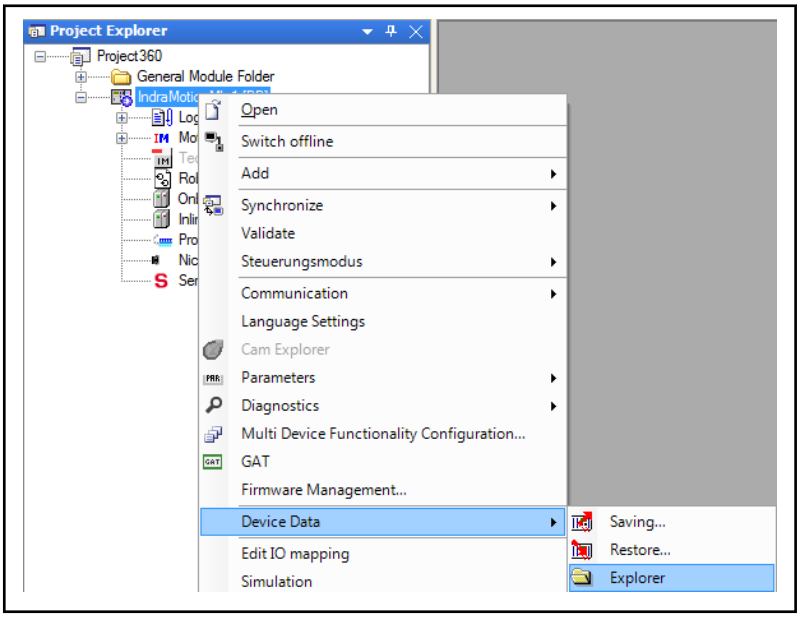


Fig. 5-8: Context menu of the control

4. Navigate to the user partition on the Compact Flash card of the control.

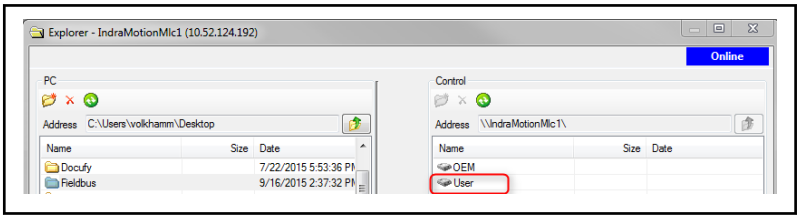


Fig. 5-9: User partition

5. Transfer of the configuration file to the user partition.

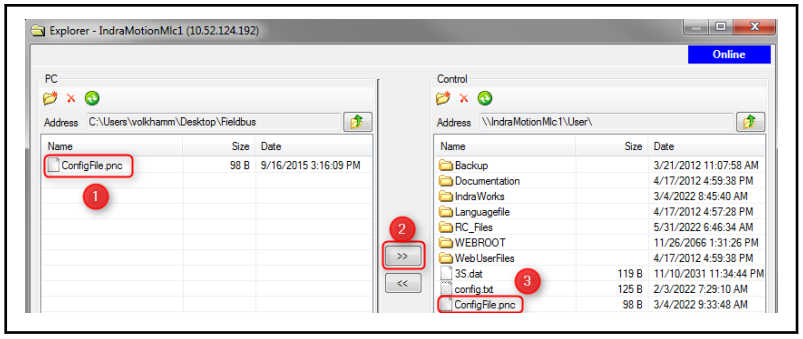


Fig. 5-10: Transferring configuration file

- Specify the name of the backup file "strBackupFileName"

  Enter the file name without extension, e.g. "Backupfile"
- Processing starts at a rising edge at "bExecute":
  - Reading the parameters specified by the configuration file of the Profinet I/O device addressed by "IdentStr"
  - Storing data on the backup file

Declaration

```
PROGRAM ProgIL_PNIODeviceBackup
VAR
    fbIL_PNIODeviceBackup: IL_PNIODeviceBackup;
    bExecute: BOOL;                              // FB running when TRUE
    strIdentString: STRING := 'Lx5-PN';          // Station name of PN
    strConfigFileName: STRING:= 'ConfigFile';    // Config file name
    strBackupFileName: STRING:= 'BackupFile';    // Backup file name
    tTimeout: TIME := T#3S;                      // Timeout monitoring of FB
    bDone: BOOL;                                 // FB completed its task
    bActive: BOOL;                               // FB working on actual task
    bError: BOOL;                                // Indicates an error
    enErrorID: RIL_COMMONTYPES.ERROR_CODE;       // Class of error
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT; // Detailed info on error
END_VAR
```

Implementation

```
fbIL_PNIODeviceBackup(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    ConfigFileName:= strConfigFileName,
    BackUpFileName:= strBackupFileName,
    Timeout:= tTimeout);
```

Error handling

The function block **IL_PNIODeviceBackup** uses the error table PROFINETIO_TABLE (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0002	"IdentStr" is invalid
COMMUNICATION_ERROR	16#0003	**Profinet IO Profinet I/O controller diagnostics** • chapter 5.9.2 "General Error Codes " on page 191 • chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	**Profinet I/O status** • chapter 5.9.4 "Special error codes, Profinet IO status " on page 193	
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller not found in the configuration table
COMMUNICATION_ERROR	16#0104	16#0000	Timeout
INPUT_INVALID_ERROR	16#3002	16#0001	"ConfigFileName" is invalid
INPUT_INVALID_ERROR	16#3002	16#0002	"BackupFileName" is invalid
ACCESS_ERROR	16#3003	16#0000	"Configuration file" could not be read
ACCESS_ERROR	16#3003	16#0001	"Configuration file" could not be opened
ACCESS_ERROR	16#3003	16#0002	"Configuration file" format invalid
ACCESS_ERROR	16#3003	16#0003	"Backup file" could not be opened
ACCESS_ERROR	16#3003	16#0004	"Backup file" cannot be read or written

Tab. 5-17: Error codes of the IL_PNIODeviceBackup

5.3.2 IL_PNIODeviceRestore

Brief description The function block **IL_PNIODeviceRestore** is used to restore parameters of a Profinet I/O device via acyclic writing from the backup file (chapter 5.2.3 "IL_PNIOWriteRecord" on page 79).

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/Backup Restore

Tab. 5-18: Reference table

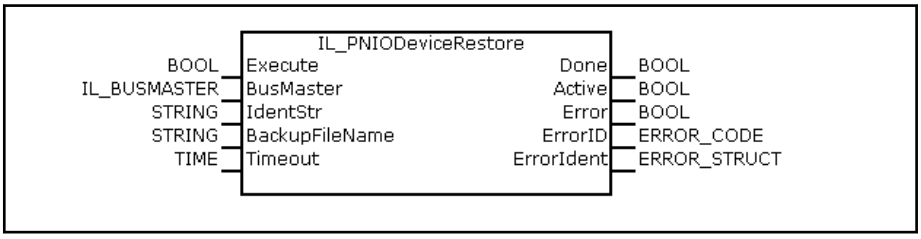


Fig. 5-11: Interface description of the IL_PNIODeviceRestore function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	IdentStr	STRING	Identification string of the Profinet I/O device
	BackupFileName	STRING	Name of backup file without extension (*.pnb)
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnostics in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE) "Error handling" on page 88

Tab. 5-19: I/O interface of the IL_PNIODeviceRestore function block

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
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
Timeout	TIME	T#0ms		T#2s	Rising edge at "Execute"

Tab. 5-20: Min./max. and default values of the IL_PNIODeviceRestore inputs

Functional description

The function block **IL_PNIODeviceRestore** is used to restore the parameters of a Profinet I/O slave device via acyclic writing from the backup file chapter 5.2.3 "IL_PNIOWriteRecord" on page 79).

The function block retrieves the parameters from the backup file of the Profinet I/O slave device stored on the Compact Flash card of the control.

The Profinet I/O device is addressed via "IdentStr".

"Slot", "SubSlot" and "Index" are used to address an individual "value" of the Profinet I/O device.

The function block retrieves all parameters from the backup file at a rising edge at "Execute" and writes the parameter values to the Profinet I/O device at the locations specified by "Slot", "SubSlot" and "Index".

Basic structure of the backup file	Example of the backup file
Slot, Sub, Slot, Index, DataSize, Data	16#0000, 16#0001, 16#0040, 16#0001, 16#58
Slot, SubSlot, Index, DataSize, Data1, Data2, Data3	16#0000, 16#0001, 16#AFF0, 16#0003, 16#20, 16#82, 16#32
...	...
...	...
...	...
Slot, SubSlot, Index, DataSize, Data1, Data2	16#0000, 16#0001, 16#AFF0, 16#0002, 16#45, 16#56



The variable at the "BackupFileName" input should only consist of the file name without extension.

The extension (*.pnb) is managed internally.

Example program

This example project describes the restoration of device parameters of the Profinet I/O device from the backup file. The backup file was saved on the Compact Flash card in the control IndraControl.

The Profinet station name "IdentStr" of the Profinet I/O device and the file name of the backup file without extension have to be specified.

The process is executed at a rising edge at the "bExecute" input: Reading the parameters from the backup file and writing to the Profinet I/O device to be identified by "strIdent".

Declaration

```
PROGRAM ProgIL_PNIODeviceRestore
VAR
    fbIL_PNIODeviceRestore: IL_PNIODeviceRestore;  // FB instance
    bExecute: BOOL;                                // FB running when TRUE
    strIdentString: STRING := 'Lx5-PN';            // Station name of PN
    strFileName: STRING:= 'BackupFile';            // Backup file name
    tTimeout: TIME := T#2S;                        // Timeout monitoring the FB
    bDone: BOOL;                                   // FB completed its task
    bActive: BOOL;                                 // FB working on actual task
    bError: BOOL;                                  // Indicates an error
    enErrorID: RIL_COMMONTYPES.ERROR_CODE;         // Class of error
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT;    // Detailed info on error
END_VAR
```

Implementation

```
fbIL_PNIODeviceRestore(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    BackUpFileName:= strFileName,
    Timeout:= tTimeout);
```

Error handling

The function block **IL_PNIODeviceRestore** uses the error table PROFINETIO_TABLE (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0002	"IdentStr" is invalid
COMMUNICATION_ERROR	16#0003	**Profinet IO Profinet I/O controller diagnostics** • chapter 5.9.2 "General Error Codes " on page 191 • chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	**Profinet I/O status** • chapter 5.9.4 "Special error codes, Profinet IO status " on page 193	
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller not found in the configuration table
COMMUNICATION_ERROR	16#0104	16#0000	Timeout
INPUT_INVALID_ERROR	16#3002	16#0002	"BackupFileName" is invalid
ACCESS_ERROR	16#3003	16#0002	"Backup file" format is invalid
ACCESS_ERROR	16#3003	16#0003	"Backup file" could not be opened
ACCESS_ERROR	16#3003	16#0004	Reading/writing "backup file" not possible

Tab. 5-21: Error codes of the IL_PNIODeviceRestore function block

5.4 Supporting the device replacement

5.4.1 IL_PNIOAssignName

Brief description The function block **IL_PNIOAssignName** assigns the device name ("IdentStr") to a Profinet I/O device (device is **renamed**). The Profinet I/O device is addressed by its MAC address.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/DeviceReplacement
MTX	RIL_ProfinetIO/POUs/DeviceReplacement

Tab. 5-22: Reference table

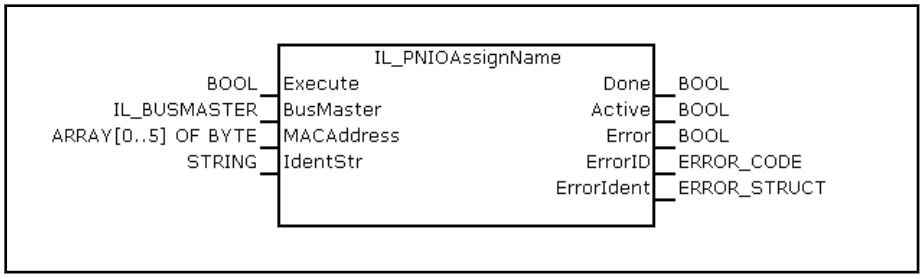


Fig. 5-12: Interface description of the IL_PNIOAssignName function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	MACAddress	ARRAY[0..5] OF BYTE	MAC address of the field bus device
	IdentStr	STRING	New identification string of the field bus device
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_Common-Types)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)

Tab. 5-23: I/O interface of the IL_PNIOAssignName function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
MACAddress	ARRAY[0..5] OF BYTE	[0,0,0,0,0,0]	-	[0,0,0,0,0,0]	Rising edge at "Execute"
IdentStr	STRING	-	-	''	Rising edge at "Execute"

Tab. 5-24: Min./max. and default values of the IL_PNIOAssignName inputs

Functional description

The function block **IL_PNIOAssignName** assigns a station name ("IdentStr") to a Profinet I/O device. The MAC address of the device is used for addressing. The MAC address is in the function block **IL_PNIOGetLiveList**. The station name "IdentStr" is a non-case sensitive string.

The station name may consist of the following characters:

- Lower case letters without umlauts "a" to "z".

  Upper case letters "A" to "Z" are allowed, but internally converted into lower case letters.

  The station names are not case sensitive.
- Digits from "0" to "9" (may not be at the beginning of the name).
- Special hyphen character "-" and dot "." (may neither be at the beginning nor at the end of the name).
- The name may not exceed 240 characters. For some devices/vendors, a limit of 127 characters apply.

  The name can be divided into substrings using "." dots. Such a substring may not exceed 63 characters.

  The name may not start with the characters string "port-xyz" (with x,y,z = 0...9).

  The name may not have the sequence "a.b.c.d" (with a,b,c,d = 0...999).

The service can only be used once per Profinet I/O controller. The Profinet I/O service DCP set is used.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	2	"IdentStr" is not set
INPUT_RANGE_ERROR	16#0001	3	"IdentStr" is invalid
INPUT_RANGE_ERROR	16#0001	4	MAC address is invalid (0)
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller is not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
COMMUNICATION_ERROR	16#0003	**Profinet I/O controller diagnostics** • Refer to chapter 5.9.2 "General Error Codes " on page 191 • Refer to chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	**Profinet I/O status** • Refer to chapter 5.9.4 "Special error codes, Profinet IO status " on page 193	

Tab. 5-25: Error codes of the IL_PNIOAssignName function block

5.4.2 IL_PNIOCheckTopology

Brief description

The function block **IL_PNIOCheckTopology** checks the saved topology of the Profinet device replacement function using the currently available Profinet topology. This is executed by reading out the port information from the Profinet I/O devices.

The function block can for example be used to detect changes in cabling or arrangement of the devices.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs

Tab. 5-26: Reference table

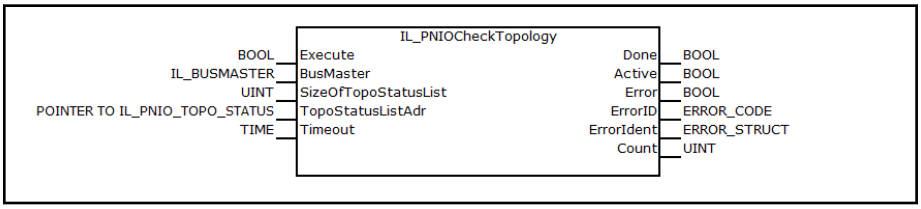


Fig. 5-13: Interface description of the IL_PNIOCheckTopology function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	SizeOfTopoStatus-List	UINT	Size of the "TopoStatusListAdr" array in bytes (e.g. use SIZEOF())
	TopoStatusListAdr	POINTER TO IL_PNIO_TOPO_STATUS	Pointer to an array of type IL_PNIO_TOPO_STATUS (e.g. use ADR()). The size of this array has to be transferred at the input "SizeOfTopoStatusList". Only devices with an error are displayed chapter 5.8.9 "IL_PNIO_TOPO_STATUS" on page 188
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnostics in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE) "Error handling" on page 96
	Count	UINT	Number of valid elements in "TopoStatusListAdr"

Tab. 5-27: I/O interface of the IL_PNIOCheckTopology function block

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
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
SizeOfTopoStatusList	UINT	0	N. def.	0	Rising edge at "Execute"
TopoStatusListAdr	POINTER TO **IL_PNIO_TOPO_STATUS**	-	-	0	Continuous
Timeout	TIME	T#0S	N. def.	T#20S	Rising edge at "Execute"

Tab. 5-28: Min./max. and default values of the IL_PNIOCheckTopology inputs

Functional description

Profinet I/O device exchange neighboring information via the "Link Layer Discovery Protocol" (LLDP), i.e. Profinet I/O devices know the "Station names" of their neighboring devices and also the ports used for cabling (own port and port of the neighbor), refer to the exemplary Profinet topology in the following figure.

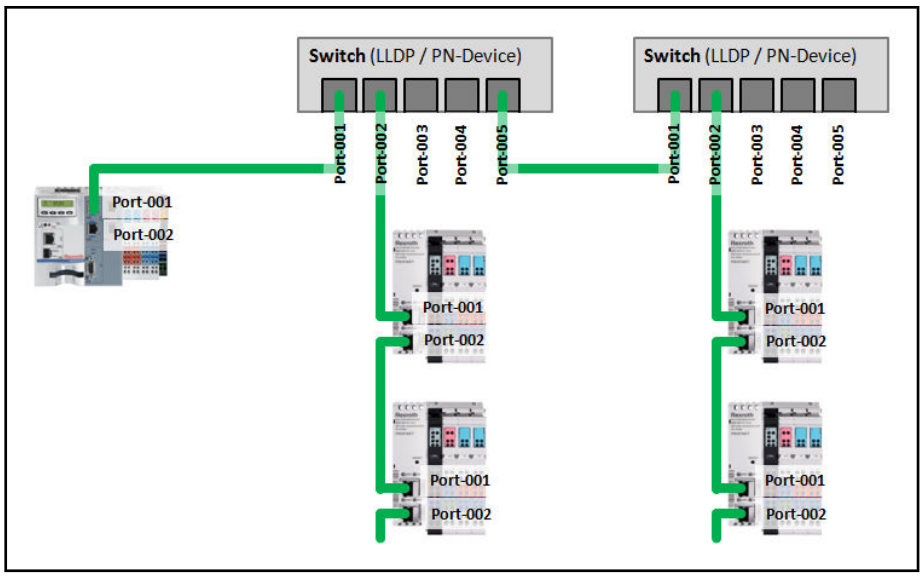


Fig. 5-14: Example Profinet topology

The Profinet I/O controller saves this neighboring information to be able to automatically name the new Profinet I/O device (same type, correct topology/cabling) with the saved station name when replacing a device.

The prerequisite is that the complete Profinet configuration is in an trouble-free communication and that the switches used support LLDP or are configured as Profinet I/O device. This is the only option for the Profinet I/O controller to read out and save the neighboring information as valid.

The function block **IL_PNIOCheckTopology** retrieves the current neighboring information from all configured Profinet I/O devices and compares them to the saved data of the device replacement function.

The station names of the neighboring devices as well as the port IDs of the respective device are compared (device ID and vendor ID are not checked).

Result is a "TopoState" for each Profinet I/O device.

Use this topology comparison for example for the following use cases:

- Check current topology or cabling with regard to whether this topology still matches the saved topology of the device replacement function. Only then, the Profinet I/O controller can automatically name when replacing a device.

  Port cabling was interchanged:

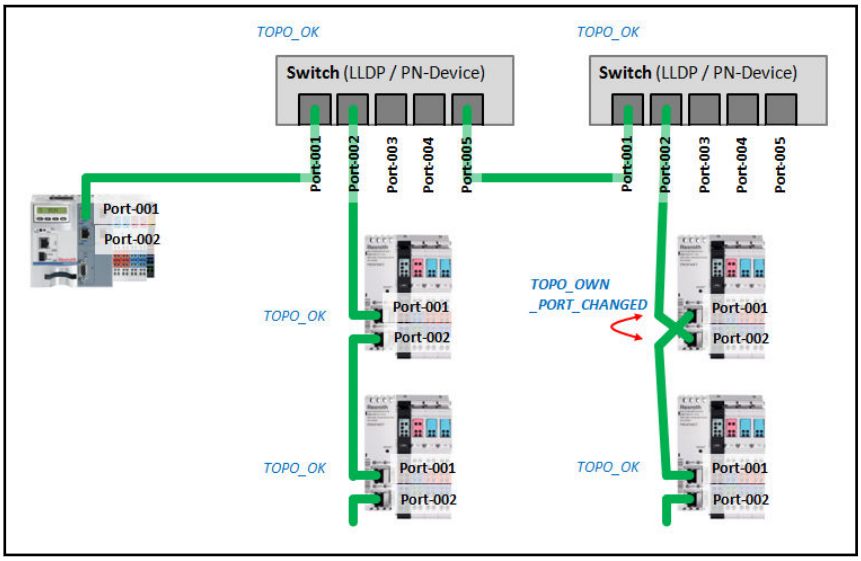


Fig. 5-15: Example Profinet topology, port cabling was interchanged

- **Cross interchange:** To check whether Profinet I/O device causes an error in a system. This Profinet I/O device is for example "cross interchanged" with an identical Profinet I/O device to check a "migration" of the error. As the Profinet I/O controller does **not** check the topology when establishing the cyclic communication, the communication would continue in case of a cross interchange even though the devices including the station name are now interchanged.

  The **IL_PNIOCheckTopology** can detect that the current neighboring information does not match anymore. Thus, the interchanged devices could now be reset in the user program ("ResetToDefault"). Thus, the device replacement function could name the reset Profinet I/O devices again according to the saved configuration.

  The figure below shows an example cross interchange. "M1Dev1" of module 1 was exchanged with "M2Dev1" of module 2.

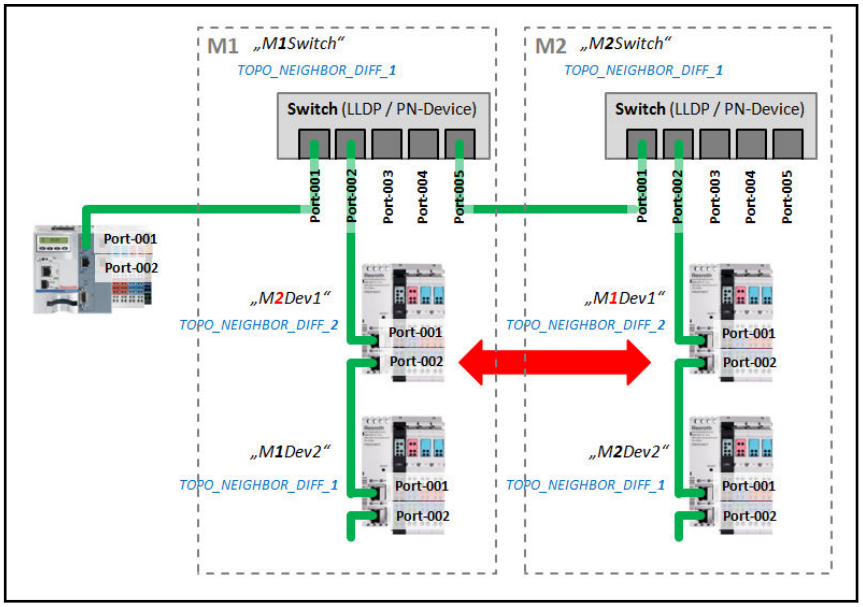


Fig. 5-16: Example Profinet topology, cross interchange

- **Finding errors and problems:** The normal Profinet I/O diagnostics already displays of a Profinet I/O cannot be reached. When comparing the topology, this error "Read-Error/Not-Reachable" would be analogously identified. The affected neighboring devices would also be identified, thus helping to find the problem.

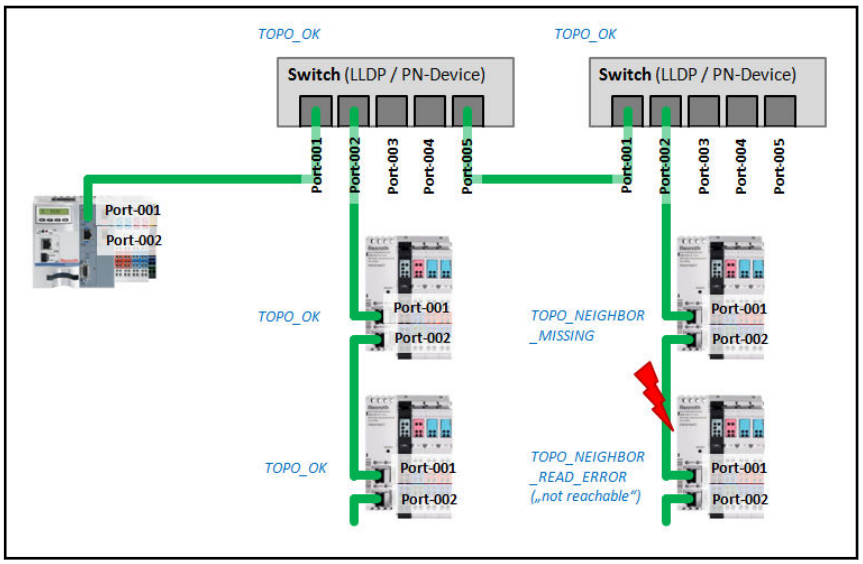


Fig. 5-17: Example Profinet topology, localization of errors and problems

Input "Timeout":

If a Profinet I/O device cannot be reached, a timeout is caused in the internal "ReadImplicit" service. This timeout is 3 s and cannot be changed externally.

If multiple Profinet I/O devices cannot be reached, set a corresponding high value at the "Timeout" input of the **IL_PNIOCheck-Topology**. Otherwise, this function block displays a "User-Timeout" error.

☞ **MAC address in the "TopoStatusList":**

The MAC addresses of the Profinet I/O devices are only read out upon the startup of the Profinet I/O controller. That means that if a device is replaced by a spare device at runtime (without restart of the Profinet I/O controller) the MAC addresses are not refreshed!

☞ **Dependency on the device replacement function:**

The prerequisite to execute the "CheckTopology" function is that the device replacement function saved the topology as valid (refer for example to "ReadyForReplace" at **IL_PNIODeviceReplacement**).

Error handling

The function block **IL_PNIOCheckTopology** uses the error table PROFINETIO_TABLE (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	"TopoStatusListAdr" is "0"
INPUT_RANGE_ERROR	16#0001	16#0001	"SizeOfTopoStatusList" is "0"
INPUT_RANGE_ERROR	16#0001	16#0101	Profinet I/O device not found (name/IP)
INPUT_RANGE_ERROR	16#0001	16#0102	Buffer too small
ACCESS_ERROR	16#0005	16#0000	No port data (number of ports is 0) could be retrieved for at least one PN device. It is assumed that neighboring data (LLDP) is not supported. The topology cannot be checked.
ACCESS_ERROR	16#0005	16#0100	Saved data is not available ("ReadyForReplace" is not set)
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller is not in configuration table
COMMUNICATION_ERROR	16#0104	Internal use	Timeout
STATE_MACHINE_ERROR	16#0105	Internal use	Faulty state (internal)

Tab. 5-29: Error codes of the IL_PNIOCheckTopology

5.4.3 IL_PNIODeviceReplacement

Brief description

The function block **IL_PNIODeviceReplacement** provides a diagnostics via the status of the automatic Device replacement function. The device replacement function can also be disabled.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs
MTX	RIL_ProfinetIO/POUs

Tab. 5-30: Reference table

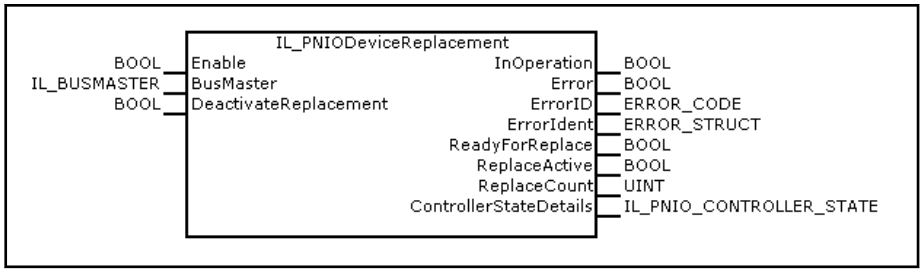


Fig. 5-18: Interface description of the IL_PNIODeviceReplacement function block

I/O type	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	DeactivateReplace-ment	BOOL	TRUE disables the automatic device replacement function of the Profinet I/O controller. If Enable=FALSE, the default state is restored, i.e. the device replacement function is enabled again
VAR_OUTPUT	InOperation	BOOL	Cyclic processing state
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	ReadyForReplace	BOOL	Indicates, whether the Profinet topology was completely detected and saved and whether the device replacement function is thus "operable"
	ReplaceActive	BOOL	Indicates that it is searched for a spare device for a configured Profinet I/O device, whose communication is not running
	ReplaceCount	UINT	Number of automatic namings by the Profinet I/O controller due to the saved topology
	StateDetails	IL_PNIO_CON-TROLLER_STATE	Detailed status information on the Profinet I/O controller stack Also refer to chapter 5.8.4 "IL_PNIO_CONTROL-LER_STATE" on page 185

Tab. 5-31: I/O interface of the IL_PNIODeviceReplacement function block

Name	Type	Min.	Max.	Default	Effective
Enable	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Enable"
DeactivateReplacement	BOOL	-	-	FALSE	Continuous

Functional description

The function block **IL_PNIODeviceReplacement** provides a diagnostics via the status of the automatic Device replacement function. The device replacement function can also be disabled.

Use the "DeactivateReplacement" input to disable the device replacement function. If `Enable=FALSE`, the default state is restored, i.e. the automatic device replacement function of the Profinet I/O controller is enabled.

The output "ReadyForReplace" indicates, whether the Profinet controller completely detected and saved the Profinet topology (prerequisite: All configured Profinet I/O devices are in trouble-free communication), i.e. the automatic device replacement function is then "operable".

"ReplaceActive" indicates that it is searched for a spare device for a configured Profinet I/O device, whose communication is not running (by checking the saved neighboring information).

Observe the following:

This flag is updated in a cycle (several seconds) slower than the Profinet diagnostics displaying the active an inactive communication of a device.

The number of automatic namings (since control switch-on) is output in "ReplaceCount".

The output "ControllerStateDetails" is identical to the function block chapter 5.5.2 "IL_PNIOControllerStateDetails" on page 116, i.e. the detailed diagnostics on the Profinet I/O controller are output. These include for example the number of configured Profinet IO devices and the number of the Profinet IO devices in active communication.

Troubleshooting

The function block **IL_PNIODeviceReplacement** uses the **error table PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
SYSTEM_ERROR	16#0005	Internal use	Internal error (RIL_ProfinetIO.library)
SYSTEM_ERROR	16#0101	Internal use	Controller not found in configuration table

Tab. 5-32: Error codes of the IL_PNIODeviceReplacement function block

5.4.4 IL_PNIODeviceResetToDefault

Brief description

The function block **IL_PNIODeviceResetToDefault** resets a Profinet I/O device to factory default. The Profinet I/O device is addressed via the MAC address.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/DeviceReplacement
MTX	RIL_ProfinetIO/POUs/DeviceReplacement

Tab. 5-33: Reference table

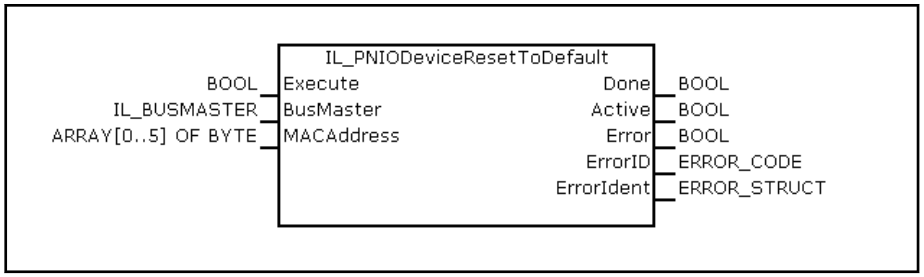


Fig. 5-19: Interface description of the IL_PNIODeviceResetToDefault function block

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	MACAddress	ARRAY[0..5] OF BYTE	MAC address of the field bus device
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_Common-Types)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)

Tab. 5-34: I/O interface of the IL_PNIODeviceResetToDefault function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”
MACAddress	ARRAY[0..5] OF BYTE	[0,0,0,0,0,0]		[0,0,0,0,0,0]	Rising edge at “Execute”

Tab. 5-35: Min./max. and default values of the IL_PNIODeviceResetToDefault inputs

Functional description

The function block **IL_PNIODeviceResetToDefault** resets a Profinet I/O device to factory default. Amongst others, the device name is reset or deleted. The Profinet I/O device is addressed via the MAC address. The MAC address is in the function block **IL_PNIOGetLiveList**.

The service can only be used once per controller.

Troubleshooting

The function block **IL_PNIODeviceResetToDefault** uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	4	MAC address is invalid (0)
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
COMMUNICATION_ERROR	16#0003	**Profinet I/O controller diagnostics** • Refer to chapter 5.9.2 "General Error Codes " on page 191 • Refer to chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	*Profinet I/O status* • Refer to chapter 5.9.4 "Special error codes, Profinet IO status " on page 193	

Tab. 5-36: Error codes of the IL_PNIODeviceResetToDefault function block

5.4.5 IL_PNIODeviceSetIP

Brief description The function block **IL_PNIODeviceSetIP** sets the IP settings of a Profinet I/O device. The Profinet I/O device is addressed via its MAC address.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/DeviceReplacement
MTX	RIL_ProfinetIO/POUs/DeviceReplacement

Tab. 5-37: Reference table

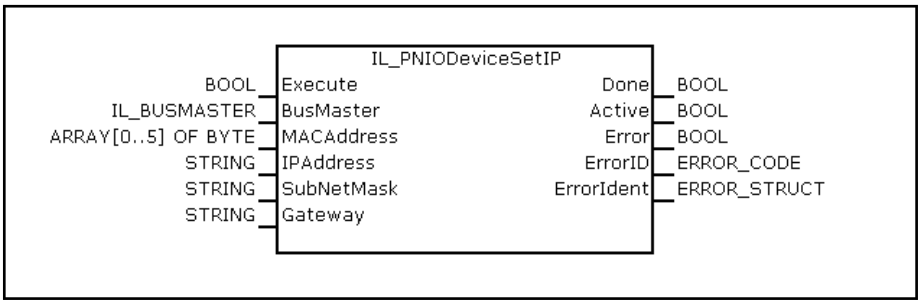


Fig. 5-20: Interface description of the IL_PNIODeviceSetIP function block

I/O type	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	MACAddress	ARRAY[0..5] OF BYTE	MAC address of the field bus device
	IPAddress	STRING(16)	IP address of the field bus device
	SubNetMask	STRING(16)	Subnet mask of the field bus device
	Gateway	STRING(16)	Gateway address of the field bus device
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_Common-Types)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)

Tab. 5-38: I/O interface of the IL_PNIODeviceSetIP function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
MACAddress	ARRAY[0..5] OF BYTE			[0,0,0,0,0,0]	Rising edge at "Execute"
IPAddress	STRING	’0.0.0.0’	’255.255.255.255’	’0.0.0.0’	Rising edge at "Execute"
SubNetMask	STRING	’0.0.0.0’	’255.255.255.255’	’0.0.0.0’	Rising edge at "Execute"
Gateway	STRING	’0.0.0.0’	’255.255.255.255’	’0.0.0.0’	Rising edge at "Execute"

Tab. 5-39: Min./max. and default values of the IL_PNIODeviceSetIP inputs

Functional description

The function block **IL_PNIODeviceSetIP** sets the IP address, the subnet mask and the gateway address of a Profinet I/O device using the DCP set service. The transfer values have to be available as "aaA.bbB.ccC.ddD" (a, b, c, d are optional) and are checked for plausibility. The Profinet I/O device is addressed via the MAC address. The MAC address is in the function block **IL_PNIOGetLiveList**. The service can only be used once per Profinet I/O controller.

Troubleshooting

The function block **IL_PNIODeviceSetIP** uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	4	MAC address is invalid (0)
INPUT_RANGE_ERROR	16#0001	5	IP address is invalid
INPUT_RANGE_ERROR	16#0001	6	Subnet mask is invalid
INPUT_RANGE_ERROR	16#0001	7	Gateway is invalid
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller is not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
COMMUNICATION_ERROR	16#0003	**Profinet I/O controller diagnostics** • Refer to chapter 5.9.2 "General Error Codes " on page 191 • Refer to chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	
COMMUNICATION_ERROR	16#0004	**Profinet I/O status** • Refer to chapter 5.9.4 "Special error codes, Profinet IO status " on page 193.	

Tab. 5-40: Error codes of the IL_PNIODeviceSetIP function block

5.4.6 IL_PNIOGetLiveList

Brief description The function block **IL_PNIOGetLiveList** provides a list of the currently reachable Profinet I/O devices.

> ☞ A cyclic call of the function block **IL_PNIOGetLiveList** can cause a malfunction of the Profinet functions, e.g. >Scan for devices< or >Device replacement<

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ DeviceReplacement
MTX	RIL_ProfinetIO/ POUs/ DeviceReplacement

Tab. 5-41: Reference table

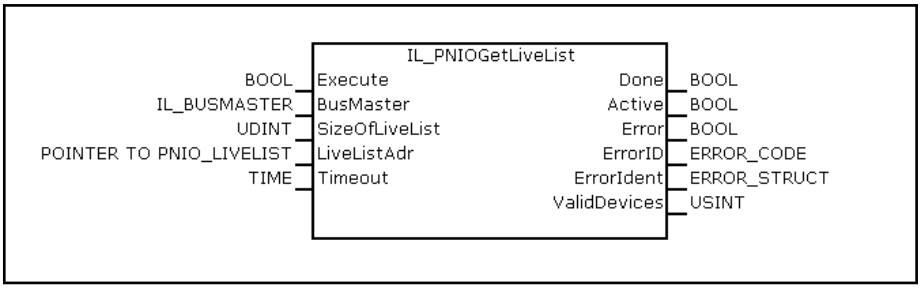


Fig. 5-21: Interface description of the IL_PNIOGetLiveList function block

I/O type	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	SizeOfLiveList	UDINT	Maximum size of the data range of "LiveListAdr" in byte. Has to be an integer multiple of SIZEOF(PNIO_LIVELIST)
	LiveListAdr	PNIO_LIVELIST	Pointer to the data filing of the live list. It should be a pointer to an array of type PNIO_LIVELIST Also refer to chapter 5.8.13 "PNIO_LIVELIST" on page 189
	Timeout	TIME	Timeout – Maximum waiting time of the Profinet I/O controller for feedback. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	ValidDevices	USINT	Number of valid Profinet I/O devices

Tab. 5-42: I/O interface of the IL_PNIOGetLiveList function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
SizeOfLiveList	UDINT	0	Not def.	0	Rising edge at "Execute"
LiveListAdr	POINTER TO PNIO_LIVELIST	Not def.	Not def.	0	Rising edge at "Execute"
Timeout	TIME	T#0s	T#60s	T#20s	Rising edge at "Execute"

Tab. 5-43: Min./max. and default values of the IL_PNIOGetLiveList inputs

Functional description

The function block **IL_PNIOGetLiveList** triggers a DCP-IdentAll service and waits for Profinet I/O device feedback during the timeout.

The feedback is entered in the list.

If not all Profinet I/O devices can be entered into the list, an error message is output.

Troubleshooting

The function block **IL_PNIOGetLiveList** uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	0	"LiveListAdr" is not set
INPUT_RANGE_ERROR	16#0001	1	"SizeOfLiveList" is not set
INPUT_RANGE_ERROR	16#0001	3	LiveListAdr: Applied list is too short
SYSTEM_ERROR	16#0101	Internal use	Profinet I/O controller is not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
COMMUNICATION_ERROR	16#0003	**Profinet I/O controller diagnostics** • Refer to chapter 5.9.2 "General Error Codes " on page 191 • Refer to chapter 5.9.3 "Special Error Codes, Profinet I/O Controller" on page 193	

Tab. 5-44: Error codes of the IL_PNIOGetLiveList function block

5.4.7 IL_PNIOGetLogMsg

Brief description

The function block gets a logger text for device exchange from the PROFINET IO controller.

The oldest text is always obtained. Each text can only be obtained once.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ DeviceReplacement
MTX	RIL_ProfinetIO/ POUs/ DeviceReplacement

Tab. 5-45: Reference table

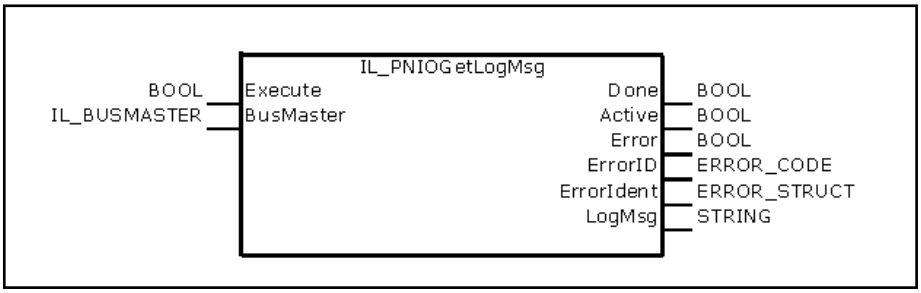


Fig. 5-22: FB IL_PNIOGetLogMsg interface

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184

	Name	Type	Comment
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_Common-Types)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	LogMsg	STRING(128)	Logger text of the device exchange

Tab. 5-46: FB IL_PNIOGetLogMsg I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”

Tab. 5-47: Minimum/maximum values and default values of the IL_PNIOGet-LogMsg inputs

Functional description

The processes are protocolled in a log file during the automatic device exchange.

By using this block, the oldest protocol message can be read once.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
STATE_MACHINE_ERROR	16#0105	0	Error in state machine

Tab. 5-48: Error codes of the IL_PNIOGetLogMsg function block

5.4.8 IL_PNIOGetPortData

Brief description

The function block **IL_PNIOGetPortData** provides port data ("neighboring information") from a Profinet I/O device.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs

Tab. 5-49: Reference table

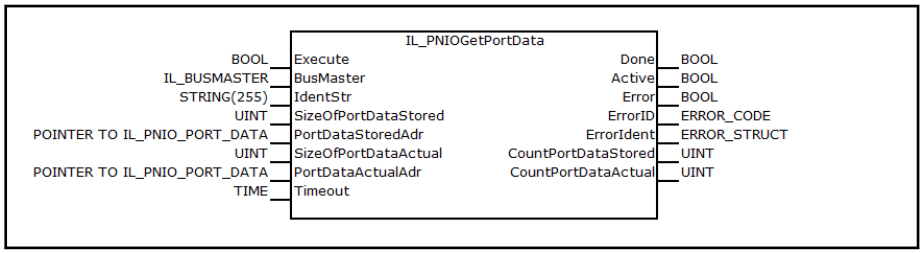


*Fig. 5-23: Interface description of the **IL_PNIOGetPortData** function block*

I/O type	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0 Also refer to chapter 5.8.1 "IL_BUSMASTER" on page 184
	IdentStr	STRING	Identification string of the Profinet I/O device
	SizeOfPortDataStored	UINT	Size of the "PortDataStoredAdr" array in bytes (e.g. use SIZEOF())
	PortDataStoredAdr	POINTER TO **IL_PNIO_PORT_DATA**	[optional] pointer (e.g. ADR()) to an array of the saved ports from "PNTopo_n.xml" (device replacement function of the Profinet I/O controllers). If this port data should not be read out, set this pointer to "0" chapter 5.8.5 "IL_PNIO_PORT_DATA" on page 186
	SizeOfPortDataActual	UINT	Size of the "PortDataActualAdr" array in bytes (e.g. use SIZEOF())
	PortDataActualAdr	POINTER TO **IL_PNIO_PORT_DATA**	[optional] Pointer (e.g. ADR() to an array of current port data of the Profinet I/O device). If this port data should not be read out, set this pointer to "0" chapter 5.8.5 "IL_PNIO_PORT_DATA" on page 186
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnostics in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE) "Error handling" on page 107
	CountPortDataStored	UINT	Number of valid elements in "PortDataStoredAdr"
	CountPortDataActual	UINT	Number of valid elements in "PortDataActualAdr"

Tab. 5-50: I/O interface of the IL_PNIOGetPortData function block

Min./max. and default values: The following table lists the min./max. and default values of the function block inputs.

Name	Type	Min.	Max.	Default	Application
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
IdentStr	STRING(255)	-	-	„“	Rising edge at "Execute"
SizeOfPortData-Stored	UINT	0	N. def.	0	Rising edge at "Execute"
PortDataStore-dAdr	POINTER TO IL_PNIO_PORT_DATA	-	-	0	Rising edge at "Execute"
SizeOfPortDa-taActual	UINT	0	N. def.	0	Rising edge at "Execute"
PortDataActua-lAdr	POINTER TO IL_PNIO_PORT_DATA	-	-	0	Rising edge at "Execute"
Timeout	TIME	T#0s	N. def.	T#2s	Rising edge at "Execute"

Tab. 5-51: Min./max. and default values of the IL_PNIOGetPortData inputs

Functional description

The function block **IL_PNIOGetPortData** provides port data ("neighboring information") from a Profinet I/O device.

To retrieve port data from a PN device, the PN device has to be provided with a valid IP address. Thus, only the current port data can be configured from a configured PN device (the controller specifies the configured IP address).

The saved port data of the device replacement function of the Profinet I/O controller can be read out and/or the current port data from the device.

For port data that is not to be read out, set the respective input to "0" ("PortDataStoredAdr"/"PortDataActualAdr").

To read out the saved port data, the device replacement function had to save the port data as valid ("ReadyForReplace" is TRUE, refer to the function block **IL_PNIODeviceReplacement**).

Error handling

The function block **IL_PNIOGetPortData** uses the error table PROFINETIO_TABLE (ERROR_TABLE = 16#0200).

The function block can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	"IdentStr" is invalid
INPUT_RANGE_ERROR	16#0001	16#0001	"PortDataStoredAdr" and "PortDataActualAdr" are **both**"0"
INPUT_RANGE_ERROR	16#0001	16#0002	"SizeOfPortDataStored" is "0"
INPUT_RANGE_ERROR	16#0001	16#0003	"SizeOfPortDataActual" is "0"
INPUT_RANGE_ERROR	16#0001	16#0101	Profinet I/O device not found (name/IP)
INPUT_RANGE_ERROR	16#0001	16#0102	Buffer too small

ErrorID	Additional1	Additional2	Remarks
ACCESS_ERROR	16#0005	16#0100	Saved data is not available ("ReadyForReplace" is not set)
ACCESS_ERROR	16#0005	Internal use	Reading error to the current Profinet I/O device (e.g. not reachable)
SYSTEM_ERROR	16#0101	Internal use	Controller not found in configuration table
COMMUNICATION_ERROR	16#0104	Internal use	Timeout
STATE_MACHINE_ERROR	16#0105	Internal use	Faulty state (internal)

Tab. 5-52: Error codes of the IL_PNIOGetPortData

5.4.9 IL_PNIOStoreTopology

Brief description The function blocks starts the generation of the current network topology data basis if all configured devices have been detected.

The topology is stored in an XML file.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ DeviceReplacement
MTX	RIL_ProfinetIO/ POUs/ DeviceReplacement

Tab. 5-53: Reference table

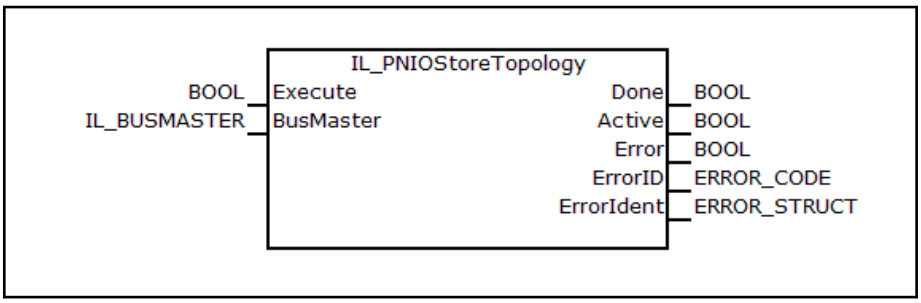


Fig. 5-24: FB IL_PNIOStoreTopology interface

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_Common-Types)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)

Tab. 5-54: FB IL_PNIOStoreTopology I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”

Tab. 5-55: Minimum/maximum values and default values of the IL_PNIOStoreTopology inputs

Functional description

The function block starts generation of the network topology data basis.

Precondition: All devices have to be free from errors. The information is saved as XML file.

The file name and the file path of the XML file can be specified via the configuration file of the control.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
STATE_MACHINE_ERROR	16#0105	0	Error in state machine
COMMUNICATION_ERROR	16#0003	*Profinet I/O controller diagnostics* • See General Error Codes , page 191, • See Special Error Codes, Profinet I/O Controller, page 193.	

Tab. 5-56: Error codes of the IL_PNIOStoreTopologie function block

5.4.10 IL_PNIORestoreTopology

Brief description

The function block IL_PNIORestoreTopology tries to restore the saved topology of the PNIO controller (e.g. use case "cross-exchange" of PN devices).

The function block can also be used to check the topology information, i.e., whether the "neighboring information" of all configured PN devices result in a completely connected network.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ DeviceReplacement
MTX	RIL_ProfinetIO/ POUs/ DeviceReplacement

Tab. 5-57: Reference table

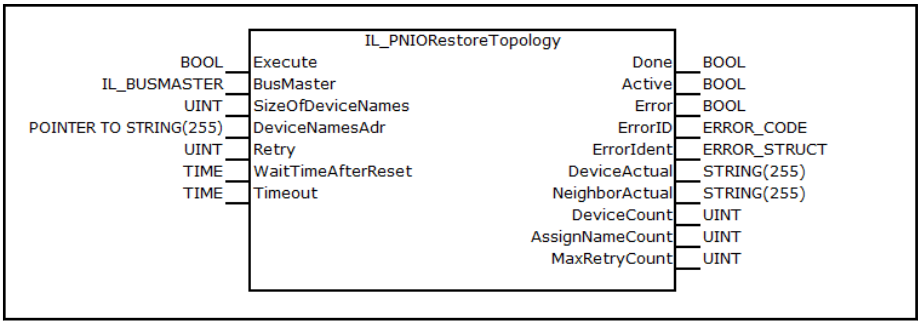


Fig. 5-25: Interface of the IL_PNIORestoreTopology function block

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	SizeOfDeviceNames	UINT	Size of array DeviceNamesAdr in bytes (e.g. SIZEOF())
	DeviceNamesAdr	POINTER TO STRING(255)	Pointer to an array of STRING(255) (e.g. ADR()). The processed device names are entered here. The size of this array has to be transferred in the parameter "SizeOfDeviceNames"
	Retry	UINT	Number of attempts for subordinate basic function calls until the function block reports an error.
	WaitTimeAfterReset	TIME	Waiting period until a PN device can be reached again after a reset (for switches up to 50 s.).
	Timeout	TIME	Timeout for subordinate basic functions, T#0s = disabled.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	DeviceActual	STRING(255)	DeviceName of the currently checked station
	NeighborActual	STRING(255)	DeviceName of the neighbor currently checked
	DeviceCount	UINT	Number of processed PN devices (refer to DeviceNamesAdr)
	AssignNameCount	UINT	Number of executed "namings" of PN devices
	MaxRetryCount	UINT	Maximum number of executions of subordinate basic functions

Tab. 5-58: I/O interface of the IL_PNIORestoreTopology function block

Name	Type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at “Execute”
SizeOfDevice-Names	UINT	0	-	0	Rising edge at “Execute”
DeviceNamesAdr	POINTER TO STRING(255)	-	-	0	Continuous
Retry	UINT	0	-	5	Rising edge at “Execute”
WaitTimeAfterRe-set	TIME	0	-	T#20s	Rising edge at “Execute”
Timeout	TIME	0	-	T#3s	Rising edge at “Execute”

Tab. 5-59: Min./max. and default values of the IL_PNIORestoreTopology inputs

Functional description

Use case: Checking the Profinet topology information

The device exchange function of the PNIO controller automatically saves the topology information ("neighboring information"/"port data") of the Profinet network.

The IL_PNIORestoreTopology function blocks runs through all devices of the PN network starting at the controller by retrieving port data and thus checks whether the neighboring data of all devices form a completely connected network.

If a switch, which is not supporting this port information, is used for example (prerequisite: LLDP – Link Layer Discovery Protocol), the PN devices "behind" the switch to not belong to a completely connected topology information anymore. In this case, the function block reports an error, as not all configured PN devices were found using neighboring data. Analogously, the device exchange function of the PNIO controllers is not fully functional as well.

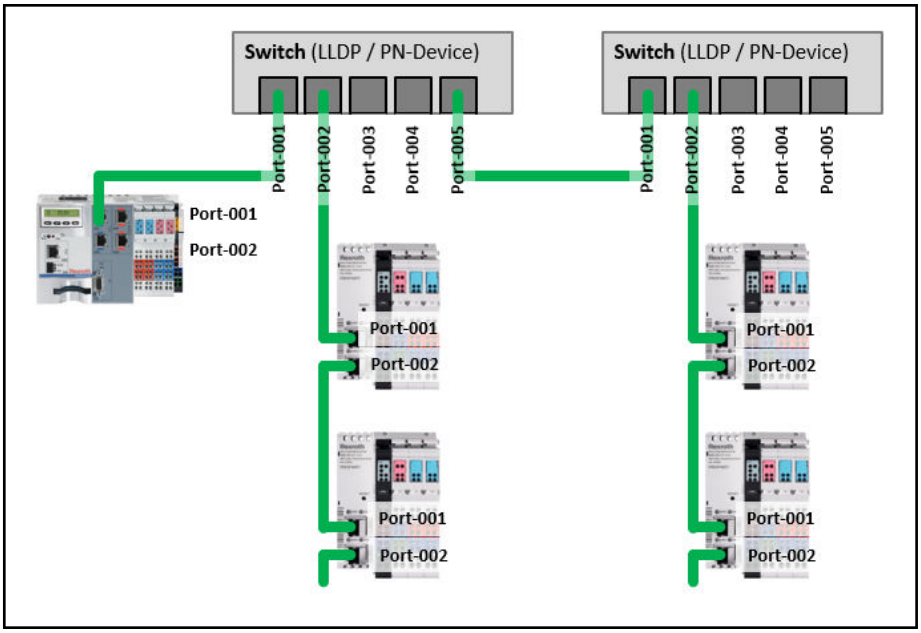


Fig. 5-26: Example: Profinet network with specified port information

Use case: Restoration of saved Profinet topology

When establishing the connection, the PNIO controller does not check the topology or the "location" at which a specific PN device is connected.

If two identical PN devices are "cross-exchanged" to check for example whether an error "moves along" at this station, each PN device keeps its saved name. That is, the topology and order of the devices would then be changed.

The IL_PNIORestoreTopology functions block runs through all devices in the network starting at the controller using neighboring information (port data) and detects the changed topology, in this case the changed "StationNames". The function block then checks the device type (VendorID and DeviceID) names the PN devices again with its saved "StationNames" to restore the topology.

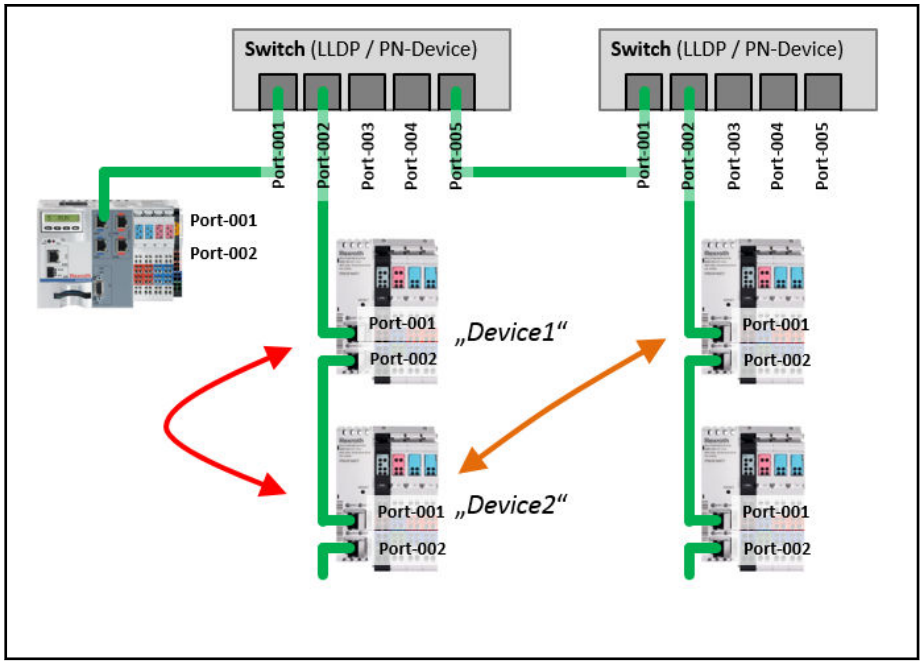


Fig. 5-27: Example: Profinet network, use case "Cross exchange"

If the device type does not match or a port slot was changed, the function block IL_PNIORestoreTopology reports a topology error. Details about topology changes can be determined using the IL_PNIOCheckTopology function block.

Notes on the parameterization and use of the function block

To be able to execute the IL_PNIORestoreTopology function block, it is required that the device exchange function of the PNIO controller saved the topology information on the Profinet network, refer to the "ReadyForReplace" output at the IL_PNIODeviceReplacement function block.

While executing the IL_PNIORestoreTopology function block, it is recommended to disable the device exchange function of the PNIO controller (refer to the IL_PNIODeviceReplacement function block), as internally used resources can block each other.

Before the IL_PNIORestoreTopology function block (re)names a PN device, a "ResetToDefault" of the PN device is executed. As PN devices can have very different startup times, e.g. some switches more than 50 s, it can be set at the "WaitTimeAfterReset" input.

Additionally, a number of repetitions of the subordinate basic functions ("ResetToDefault", "AssignName", "DCPGet") can be parameterized at the "Retry" input to increase the "robustness" of the IL_PNIORestoreTopology function block. The product from "WaitTimeAfterReset" and "Retry" thus increases the availability of the function. However, the execution time of the function block, e.g. also the time until the function block reports an error due to a change in topology, increases at the same time.

The function block IL_PNIORestoreTopology can only name a PN device correctly again or can only restore the topology, if the PN devices used in the cyclic operation support the DCP service "Reset to factory settings" (or "Reset to default").

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00000001	16#00000000	“DeviceNamesAdr” input is 0
INPUT_INVALID_ERROR	16#00000001	16#00000001	“SizeOfDeviceNames” input is 0
INPUT_RANGE_ERROR	16#00000001	16#00000003	List assigned to DeviceNamesAdr is too short
INPUT_INVALID_ERROR	16#00000001	x	See also IL_PNIOGetPortData, page 105
ACCESS_ERROR	16#00000005	16#00000100	Saved data is not complete (ReadyForReplace is not set)
ACCESS_ERROR	16#00000005	Internal	Reading error, current device (possible that it cannot be reached)
INPUT_RANGE_ERROR	16#00000009	Internal	Max. number of permitted devices exceeded
ACCESS_ERROR	16#00000030	Internal	Retry exceeded: access/com. error, suggestion: Increase Retry/WaitTime/Timeout
ACCESS_ERROR	16#00000031	Internal	No all configured devices found (maybe not reachable)
OTHER_ERROR	16#00000032	16#00000000	Topology error: Port interchanged (or neighboring device missing)
OTHER_ERROR	16#00000032	16#00000001	Topology error: Different device type (VendorID / DeviceID)
OTHER_ERROR	16#00000032	16#00000002	Topology error: Neighboring device missing (or ports interchanged)
SYSTEM_ERROR	16#00000101	Internal	Controller not found in configuration tree
COMMUNICATION_ERROR	16#00000104	Internal	User timeout
STATE_MACHINE_ERROR	16#00000105	Internal	State machine error (internal)
SYSTEM_ERROR	16#00000106	Internal	Configured device not found (internal)

Tab. 5-60: Error codes of the IL_PNIORestoreTopologie function block

5.5 Diagnostics

5.5.1 IL_PNIOControllerState

Brief description This function block returns a basic diagnosis of the PROFINET I/O controller. If a diagnostic message has been reported, additional function blocks can be used to read details on the diagnostic messages (e.g., IL_PNIOControllerStateDetails on page 116 or IL_PNIORemoteDeviceState on page 126).

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ Diagnostics
MTX	RIL_ProfinetIO/ POUs/ Diagnostics

Tab. 5-61: Reference table

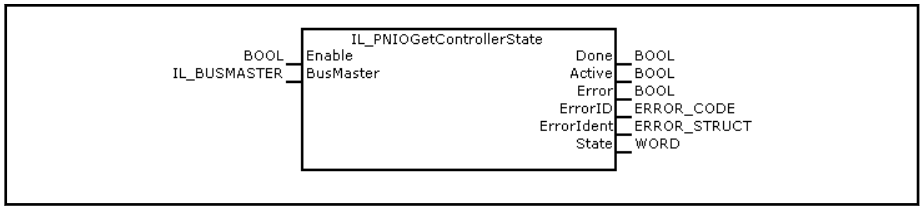


Fig. 5-28: FB IL_PNIOControllerState interface

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	State	WORD	State of the PROFINET I/O controller, see below.

Tab. 5-62: FB IL_PNIOControllerState I/O interface

The "State" output provides a rough diagnostic classification in binary code.

The bits are contained in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Output "State" (WORD)

Bit no.	Name	Description
0	IL_FBM_BUS_OFF	Field bus not ready
1	IL_FBM_MASTER_FAILURE	Bus master failure
2	IL_FBM_SLAVE_NOT_REACHABLE	One or more field bus slaves cannot be reached (slave not reachable)
3	IL_FBM_SLAVE_DIAGNOSIS	One or more field bus slaves report diagnostics (slave diag)
4	IL_FBM_SLAVE_ERROR	One or more field bus slaves report errors (slave error)
5	IL_FBM_TOPOLOGY_INVALID	Topology data invalid This is a topology data status of the device replacement function of the PNIO controller. During a PLC download or a PLC reset (cold/warm), the PNIO controller deletes the saved topology data. The topology data are required for the device replacement function. They are automatically (re-)generated and saved if all configured devices communicate correctly and no diagnostics is reported.

Tab. 5-63: Elements of State output

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at Enable

Tab. 5-64: Minimum / maximum values and default values of the IL_PNIOControllerState inputs

Functional description

The function block retrieves state information about a PROFINET controller and returns it as bit array in "State". The function block is completely processed in one PLC cycle.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
COMMUNICATION_ERROR	16#0003	*Profinet I/O controller diagnostics* • See General Error Codes , page 191, • See Special Error Codes, Profinet I/O Controller, page 193.	

Tab. 5-65: FB IL_PNIOControllerState error codes

5.5.2 IL_PNIOControllerStateDetails

Brief description

This function block determines the detailed state of the Profinet controller stack. It provides information on

- state of the communication stack,
- the state of the bus communication,
- error counters,
- an overview on configured, active slaves reporting a diagnostics

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ Diagnostics
MTX	RIL_ProfinetIO/ POUs/ Diagnostics

Tab. 5-66: Reference table

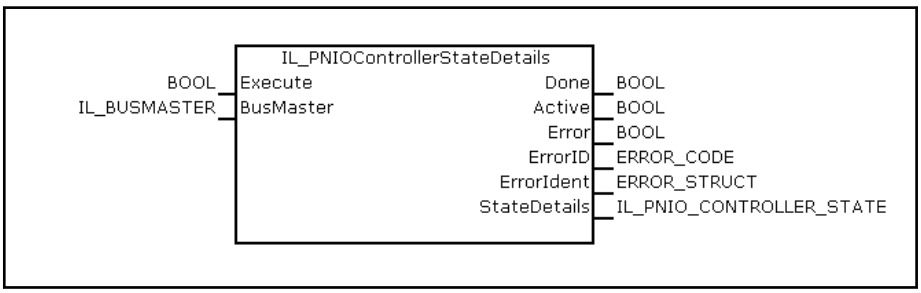


Fig. 5-29: FB IL_PNIOControllerStateDetails interface

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_Common-Types)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	StateDetails	IL_PNIO_CON-TROLLER_STATE	Detailed information about the state of the controller stack Also refer to IL_PNIO_CONTROLLER_STATE, page 185

Tab. 5-67: FB IL_PNIOControllerStateDetails I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”

Tab. 5-68: Minimum / maximum values and default values of the IL_PNIOControllerStateDetails inputs

Functional description

The function block provides current state information of the PROFINET controller.

The comparison of the following values is of particular interest for evaluation of the bus system: number of configured devices (NumOfConfigSlaves), number of devices present at the PROFINET (NumOfActiveSlaves) and number of devices with present diagnostic message (NumOfDiagSlaves).

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
COMMUNICATION_ERROR	16#0003	*Profinet I/O controller diagnostics* • See General Error Codes , page 191, • See Special Error Codes, Profinet I/O Controller, page 193.	

Tab. 5-69: FB IL_PNIOControllerStateDetails error codes

5.5.3 IL_PNIODiagnosis

Brief description

The function block IL_PNIODiagnosis creates a comprehensive diagnostics of a Profinet I/O controller and of its connected devices.

Simple diagnostics are cyclically prepared. Detailed diagnostics are acyclically and subsequently retrieved within the function block.

The diagnostics are provided in data structures as well as in "strings". It is also possible to retrieve manufacturer-specific diagnostic texts from the device description file (GSDML).

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/Diagnostics
MTX	RIL_ProfinetIO/POUs/Diagnostics

Tab. 5-70: Reference table

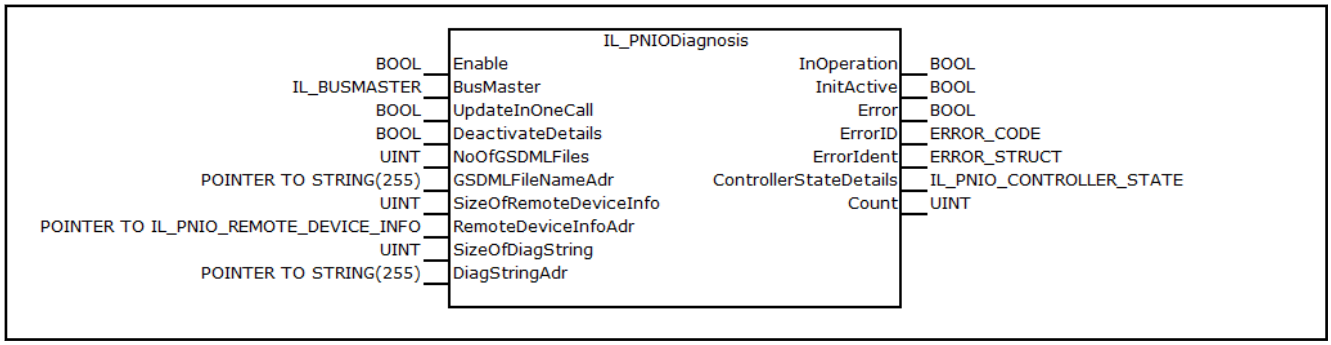


Fig. 5-30: Interface of the IL_PNIODiagnosis function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one Profinet IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	UpdateInOneCall	BOOL	If TRUE, the "DeviceState" of all devices is updated in one call, otherwise in subsequent calls (runtime optimized)
	DeactivateDetails	BOOL	Disables the acyclic reading of the detailed ”DiagInfo” in RemoteDeviceInfo
	NoOfGSDMLFiles	UINT	[optional] number of GSDML files in GSDMLFileNameAdr
	GSDMLFileNameAdr	POINTER TO STRING(255)	[optional] pointer (ADR()) to an array of STRING(255): File name of the GSDML files, incl. the path (on the Flash card of the control) if manufacturer-specific diagnostic texts should be retrieved
	SizeOfRemoteDeviceInfo	UINT	Size of the “RemoteDeviceInfoAdr” array in bytes (SIZEOF())

	Name	Type	Comment
	RemoteDeviceInfoAdr	POINTER TO IL_PNIO_REMOTE_DEVICE_INFO	Pointer (ADR()) to an array of IL_PNIO_REMOTE_DEVICE_INFO: The complete diagnostic information on one device is entered chapter 5.8.6 "IL_PNIO_REMOTE_DEVICE_INFO" on page 186
	SizeOfDiagString	UINT	[optional] Size of the “DiagStringAdr” array in bytes (SIZEOF())
	DiagStringAdr	POINTER TO STRING(255)	[optional] Pointer (ADR()) to an array of STRING(255): A diagnostic string is created for each device
VAR_OUTPUT	InOperation	BOOL	Cyclic processing state
	InitActive	BOOL	Initialization active: This can take several seconds for GSDML files with multiple thousands of lines
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	ControllerStateDetails	IL_PNIO_CONTROLLER_STATE	Detailed information about the state of the controller stack Also refer to IL_PNIO_CONTROLLER_STATE, page 185
	Count	UINT	Number of existing devices (concerns DiagStringAdr and RemoteDeviceInfoAdr)

Tab. 5-71: I/O interface of the IL_PNIODiagnosis function block

Name	Type	Min.	Max.	Default	Effective
Enable	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at Enable
UpdateInOneCall	BOOL			FALSE	Continuous
DeactivateDetails	BOOL	-	-	FALSE	Continuous
NoOfGSDMLFiles	UINT	0	N. def.	0	Rising edge at Enable
GSDMLFileNameAdr	POINTER TO STRING(255)	-	-	0	Continuous
SizeOfRemoteDeviceInfo	UINT	0	N. def.	0	Rising edge at Enable
RemoteDeviceInfoAdr	POINTER TO IL_PNIO_REMOTE_DEVICE_INFO	-	-	0	Continuous
SizeOfDiagString	UINT	0	N. def.	0	Continuous
DiagStringAdr	POINTER TO STRING(255)	-	-	0	Continuous

Tab. 5-72: Min./max. and default values of the IL_PNIODiagnosis inputs

Functional description

The function block IL_PNIODiagnosis creates a comprehensive diagnostics of a Profinet I/O controller and of its connected devices. The controller is addressed via the "BusMaster" input, the respective PN devices are automatically retrieved.

The function block operates cyclically, i.e. simple diagnostics ("DeviceState") are cyclically prepared, detailed diagnostics are acyclically and subsequently retrieved ("DiagInfo").

Use the "DeactivateDetails" input to also disable the acyclic reading of the detailed information, e.g. if other acyclic services are to be executed at a device that could otherwise be blocked.

The inputs "RemoteDeviceInfoAdr" and "DiagStringAdr" have to have (at least) the same number of array elements as the number of configured PN devices. Both arrays list the PN devices according to the order in the configuration tree.

	TestProg.arDiagStrings[INDEX]
0	ugw-micro-1 : is OK
1	### slc-0-gpnt2 : Diag Slot 0 SubSlot 0 Channel 1, ErrorType=216 TextId="T_ErrText0216" Value="Input 1 external test signal failure. Check for stuck-at-high or cabling."
2	### slc-0-gpnt : Diag Slot 0 SubSlot 0 Channel 1, ErrorType=216 TextId="T_ErrText0216" Value="Input 1 external test signal failure. Check for stuck-at-high or cabling."
3	il-pn-bk-fwxxx : is OK
4	### lx5-pn-device : not reachable
5	### il-pn-bk-fw330 : Diag Slot 2 SubSlot 0 Channel 32768, ErrorType=21008 Value="The specified device indicates a peripheral fault." TextId="ID0x5210"
6	

Fig. 5-31: Example: DiagString

Manufacturer-specific diagnostic texts in the device description (GSDML):

The "GSDMLFileNameAdr" input can be used optionally, no or only a subset of the GSDML files used can be specified.

Some devices do not have any specific diagnostic texts in the GSDML. In this case, the <ChannelDiagList> element is not part of the GSDML and using the file at the IL_PNIODiagnosis function block is of no use.

Specify the file names incl. the path (on the Flash card of the control) at the "GSDMLFileNameAdr" input.

The GSDML files have to be stored manually on the Flash card of the control. The following example shows a GSDML subfolder on the USER partition ("/ata0b/"):

```
arGSDMLFileName : ARRAY[0..2] OF STRING(255) := [
    '/ata0b/GSDML/GSDML-V2.31-MBS-MICRO3004-20150206.xml',
    '/ata0b/GSDML/GSDML-V2.1-Bosch Rexroth AG-SLC_0_GPNT-20101215.xml',
    '/ata0b/GSDML/GSDML-V2.3-Bosch Rexroth AG-R_IL PN BK DI8 DO4-20140915.xml'
];
```

When initializing the IL_PNIODiagnosis function block, the specified GSDML files are assigned to the configured PN devices.

It is assigned via the VendorID and the DeviceID.

In addition, an offset is already saved to the <ChannelDiagList> during the initialization.

As GSDML files can contain thousands of lines, retrieving information can take up to several seconds. If required, improve this manually by deleting unrequired descriptions for diagnostic texts in the GSDML.

There have to be the following keywords in the GSDML:

```
...
> could be deleted <
      ...
```

```
    <DeviceIdentity VendorID="0x011f" DeviceID="0x2403">
      ...
    </DeviceIdentity>
      ...
> could be deleted <
      ...
      <ChannelDiagList>
        <ChannelDiagItem ErrorType="0101">
          <Name TextId="T_ErrText0101" />
          <Help TextId="T_HelpChanSlot" />
        </ChannelDiagItem>
      ...
      </ChannelDiagList>
      ...
> could be deleted <
      ...
      <ExternalTextList>
        <PrimaryLanguage>
          <!--Identification-->
          <Text TextId="T_ErrText0101" Value="Internal Error:..." />
        ...
        </PrimaryLanguage>
        <Language xml:lang="de">
          <Text TextId="T_ErrText0101" Value="Interner Fehler:... />
        ...
        </Language>
      </ExternalTextList>
```

Mode of operation of the function block:

It is browsed for "ErrorType" in the <ChannelDiagList> element. "ErrorType" corresponds to the "ChannelErrorType" in the detailed diagnostics PNIO_DIAGINFO.

A "TextId" is assigned to the "ErrorType". A description text or a diagnostic text is stored in the <ExternalTextList> element for the "TextId".

The function block uses the "TextId" found first. This is normally in the PrimaryLanguage ("en"). If another language, e.g. "de" is preferred and this language is provided in the GSDML, change the sequence of the languages in the <ExternalTextList>. The "TextIds" are identical and the function block always uses the first entry found.

Example in ST *Declaration*

```
VAR
    fbIL_PNIODiagnosis: IL_PNIODiagnosis;
    boEnable: BOOL := FALSE;
    arRemoteDeviceInfo: ARRAY [0..127] OF IL_PNIO_REMOTE_DEVICE_INFO;
    DiagStrings: ARRAY [0..127] OF STRING(255);
    rControllerStateDetails: IL_PNIO_CONTROLLER_STATE;
    CountOfDevices: UINT;
    // ### Optional: name here the path of the GSDML files,
    //               which you have stored on the compact flash of the control
    //               and then assign the number of the files to the variable 'NoOfGSDMLFiles' (see below)
    arGSDMLFileName : ARRAY[0..3] OF STRING(255) := [
        // example: 'User-Partition' (ata0b) with folder '/GSDML/':
        '/ata0b/GSDML/GSDML-V2.31-MBS-MICRO3004-20150206.xml',
        '/ata0b/GSDML/GSDML-V2.1-Bosch Rexroth AG-SLC_0_GPNT-20101215.xml',
        '/ata0b/GSDML/GSDML-V2.3-Bosch Rexroth AG-R_IL PN BK DI8 DO4-20140915.xml' ,
        '/ata0b/GSDML/GSDML-V2.3-Bosch_Rexroth_AG-S20_PN_BK_plus-20150910.xml'
    ];
    NoOfGSDMLFiles : UINT := 0; // := SIZEOF(arGSDMLFileName) / SIZEOF(STRING(255));
END_VAR
```

Implementation

```
fbIL_PNIODiagnosis(
    Enable:= boEnable,
    BusMaster:= ,
    UpdateInOneCall:= ,
    DeactivateDetails:= ,
    NoOfGSDMLFiles:= NoOfGSDMLFiles,
    GSDMLFileNameAdr:= ADR(arGSDMLFileName),
    SizeOfRemoteDeviceInfo:= SIZEOF(arRemoteDeviceInfo),
    RemoteDeviceInfoAdr:= ADR(arRemoteDeviceInfo),
```

```
SizeOfDiagString:= SIZEOF(DiagStrings),
DiagStringAdr:= ADR(DiagStrings),
InOperation=> ,
Error=> ,
ErrorID=> ,
ErrorIdent=> ,
InitActive=> ,
ControllerStateDetails=> rControllerStateDetails,
Count=> CountOfDevices);
```

Troubleshooting The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Note
INPUT_INVALID_ERROR	16#0001	16#0010	“DiagString” and/or “RemoteDeviceInfo” too short
INPUT_INVALID_ERROR	16#0001	16#0011	“DiagStringAdr” is 0
INPUT_INVALID_ERROR	16#0001	16#0013	“RemoteDeviceInfoAdr” is 0
INPUT_INVALID_ERROR	16#0001	16#0014	“SizeOfRemoteDeviceInfo” is 0
INPUT_INVALID_ERROR	16#0001	16#0015	“GSDMLFileNameAdr” is 0
ACCESS_ERROR	16#0005	Internal use	Internal error
INPUT_RANGE_ERROR	16#0009	Internal use	Max. number of permitted configurable devices exceeded.
RESOURCE_ERROR	16#0010	Index 0..(n-1) of GSDMLFile-NameAdr	GSDML 'File Open Error'
RESOURCE_ERROR	16#0011	Internal use	GSDML 'File Read Error'
RESOURCE_ERROR	16#0012	Internal use	GSDML 'File Close Error'
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table

Tab. 5-73: Error codes of the IL_PNIODiagnosis function block

5.5.4 IL_PNIOGetConfigDeviceNameList

Brief description The function block reads the names of all configured PROFINET IO devices.

The user has to provide an ARRAY OF STRING(255) to accept the device names.

These device names can be used to activate the input "IdentStr" for other PNIO function blocks.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ Diagnostics
MTX	RIL_ProfinetIO/ POUs/ Diagnostics

Tab. 5-74: Reference table

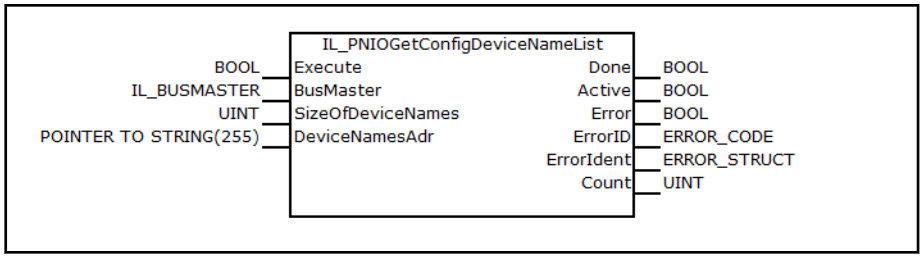


Fig. 5-32: IL_PNIOGetConfigDeviceNameList

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the field bus master in the order of the configuration tree Also refer to IL_BUSMASTER on page 184
	SizeOfDeviceNames	UINT	Size of array DeviceNamesAdr in bytes (SIZEOF())
	DeviceNamesAdr	POINTER TO STRING(255)	Pointer (ADR()) to an array of STRING(255). where the configured device names are entered. The size of this array has to be transferred in the parameter "SizeOfDeviceNames"
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	Count	UINT	Number of valid device names in DeviceNamesAdr

Tab. 5-75: Function block IL_PNIOGetConfigDeviceNameList

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”
SizeOfDeviceNames	UINT	0	N. def.	0	Rising edge at “Execute”
DeviceNamesAdr	POINTER TO STRING(255)	-	-	0	Rising edge at “Execute”

Tab. 5-76: Min./max. values and default values of the L_PNIOGetConfigDeviceNameList inputs

Functional description

The function block searches the opened IO configuration for the devices which are located below the selected PROFINET controller.

Deactivated devices are not included in the list. These device names can be used as "IdentStr" input parameters for other function blocks.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0001	0	DeviceNamesAdr: Applied list is too short
INPUT_RANGE_ERROR	16#0001	1	DeviceNamesAdr not set
INPUT_RANGE_ERROR	16#0001	2	SizeOfDeviceNames not set
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table

Tab. 5-77: Error codes of the IL_PNIOGetConfigDeviceNameList function block

5.5.5 IL_PNIOGetControllerList

Brief description The function block IL_PNIOGetControllerList provides the number of configured Profinet I/O controllers.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ Diagnostics
MTX	RIL_ProfinetIO/ POUs/ Diagnostics

Tab. 5-78: Reference table

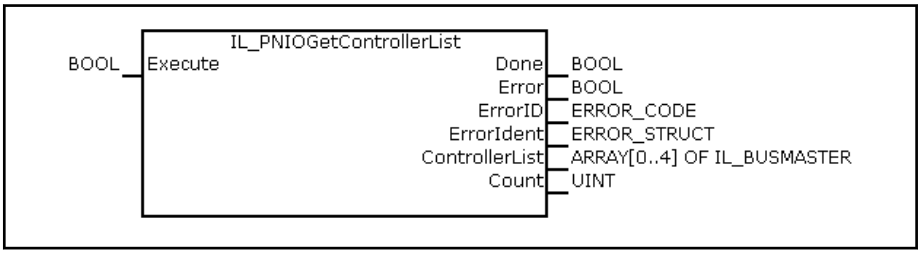


Fig. 5-33: IL_PNIOGetControllerList

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	ControllerList	ARRAY[0..4] OF IL_BUSMASTER	Array of type IL_BUSMASTER outputting the configured BusMaster (PNIO controller) Also refer to IL_BUSMASTER on page 184
	Count	UINT	Number of valid elements in the "ControllerList" output

Tab. 5-79: IL_PNIOGetControllerList function block

Name	Type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous

Tab. 5-80: Min./max. and default values of the IL_PNIOGetControllerList inputs

Functional description

The function block IL_PNIOGetControllerList provides the number of configured Profinet I/O controllers.

The "ControllerList" output (of type IL_BUSMASTER) can be used at the inputs of the other Profinet diagnostic function blocks of the RIL_ProfinetIO library.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	Internal use	Internal error (null pointer)

Tab. 5-81: Error codes of the IL_PNIOGetControllerList function block

5.5.6 IL_PNIOGetDiagDeviceNameList

Brief description

This function block reads the names of all PROFINET IO devices that deliver a diagnostic message.

The user has to provide an ARRAY OF STRING(255) to accept the device names.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ Diagnostics
MTX	RIL_ProfinetIO/ POUs/ Diagnostics

Tab. 5-82: Reference table

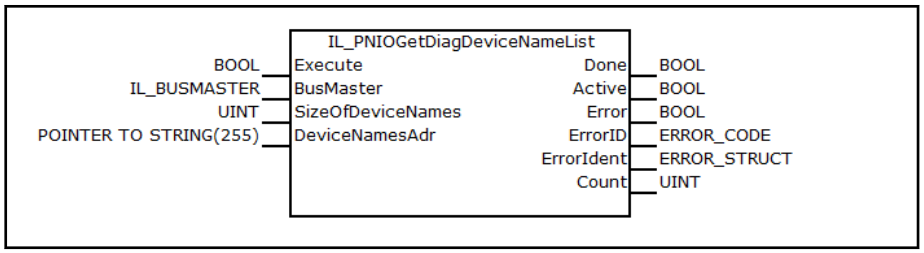


Fig. 5-34: IL_PNIOGetDiagDeviceNameList

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the field bus master in the order of the configuration tree Also refer to IL_BUSMASTER on page 184
	SizeOfDeviceNames	UINT	Size of array DeviceNamesAdr in bytes (SIZEOF())
	DeviceNamesAdr	POINTER TO STRING(255)	Pointer (ADR()) to an array of STRING(255). where the DeviceNames of the devices which signal diagnosis are entered. The size of this array must be transferred in parameter "SizeOfDeviceNames".
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid

	Name	Type	Comment
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	Count	UINT	Number of devices with active diagnosis in DeviceNamesAdr

Tab. 5-83: Function block IL_PNIOGetDiagDeviceNameList

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”
SizeOfDeviceNames	UINT	0	N. def.	0	Rising edge at “Execute”
DeviceNamesAdr	POINTER TO STRING(255)	-	-	0	Rising edge at “Execute”

Tab. 5-84: Min./max. values and default values of the IL_PNIOGetDiagDeviceNameList inputs

Functional description

All PROFINET devices currently reporting diagnostics are determined from the configuration table.

The device name is transferred.



This list does not include any device with other errors.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0001	0	DeviceNamesAdr: Applied list is too short
INPUT_RANGE_ERROR	16#0001	1	DeviceNamesAdr not set
INPUT_RANGE_ERROR	16#0001	2	SizeOfDeviceNames not set
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table

Tab. 5-85: Error codes of the IL_PNIOGetDiagDeviceNameList function block

5.5.7 IL_PNIORemoteDeviceState

Brief description

This function block determines the basic state of a device connected to the controller.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ Diagnostics
MTX	RIL_ProfinetIO/ POUs/ Diagnostics

Tab. 5-86: Reference table

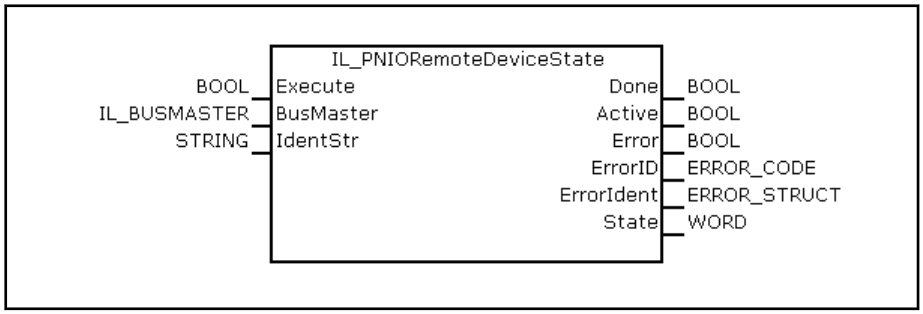


Fig. 5-35: IL_PNIORemoteDeviceState

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Enabling the function; an enabled function block cannot be interrupted.
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. Allowed values 0..4; default: 0 Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Station name of the IO device to be accessed.
VAR_OUTPUT	Done	BOOL	Completed message (successful)
	Active	BOOL	Activity display
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	State	WORD	State information of the slave at the field bus, see below

Tab. 5-87: Function block IL_PNIORemoteDeviceState

Output "State" (WORD)

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Device not existent (bus slave state – slave not existent)
1	IL_RBS_SLAVE_NOT_READY	Device not ready (bus slave state – slave not ready)
2	IL_RBS_SLAVE_CONFIG_ERROR	Device configuration error (bus slave state – slave configuration error)
3	IL_RBS_SLAVE_DIAGNOSIS	Device diagnostics available (bus slave state – slave diagnoses available)
4	IL_FBS_SLAVE_ERROR	General device error

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”
IdentStr	STRING			’’	Rising edge at “Execute”

Tab. 5-88: Minimum / maximum values and default values of the IL_PNIOControllerState inputs

Functional description The bit-coded state bar of the PROFINET device is evaluated. The IL_PNIORemoteDeviceStateDetails service can be used for further information. The device is selected by specifying the PROFINET device in the "IdentStr" variable. The devices must be configured and active so that they can be observed.

Troubleshooting The function block uses error table PROFINETIO_TABLE (ErrorIdent.Table = 16#0200). It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0001	1	IdentStr is not specified
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
INPUT_RANGE_ERROR	16#0106	0	Device “IdentStr” is not configured

Tab. 5-89: Error codes of the IL_PNIORemoteDeviceState function block

5.5.8 IL_PNIORemoteDeviceStateDetails

Brief description The function block determines the detailed PROFINET IO state of a device configured and connected to the controller.

The PROFINET data structures are summarized in general diagnostic information.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ Diagnostics
MTX	RIL_ProfinetIO/ POUs/ Diagnostics

Tab. 5-90: Reference table

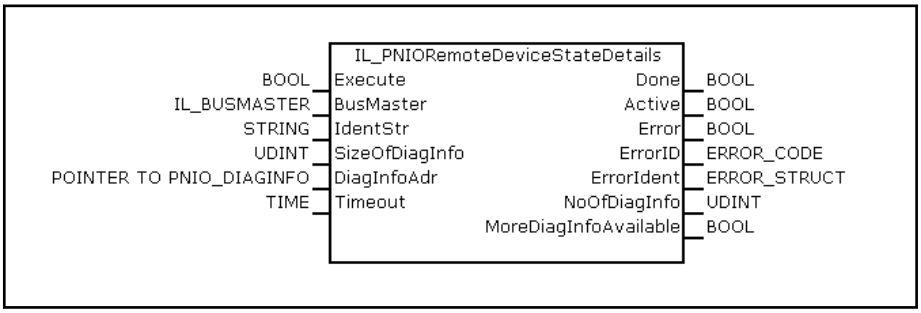


Fig. 5-36: Interface of FB IL_PNIORemoteDeviceStateDetails

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the field bus device
	SizeOfDiagInfo	UDINT	Maximum size of the data range of "DiagInfoAdr". Must be an integer multiple of SIZEOF(PNIO_DIAGINFO).
	DiagInfoAdr	POINTER TO PNIO_DIAGINFO	Points to the data storage for the evaluated diagnostic data of the device. Also refer to PNIO_DIAGINFO, page 189
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_Common-Types)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)
	NoOfDiagInfo	UDINT	Number of evaluations entered in DiagInfoAdr
	MoreDiagInfoAvaila-ble	BOOL	There is additional diagnostic information that does not have any place in the data structure provided.

Tab. 5-91: FB IL_PNIORemoteDeviceStateDetails I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at “Execute”
IdentStr	STRING			’’	Rising edge at “Execute”
SizeOfDiagInfo	UDINT	0	N. def.	0	Rising edge at “Execute”
DiagInfoAdr	POINTER TO PNIO_DIAGINFO			0	Rising edge at “Execute”
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at “Execute”

Tab. 5-92: Minimum / maximum values and default values of the IL_PNIOControllerStateDetails inputs

Functional description The state of the device is determined by acyclic read accesses.

The number of accesses required may vary depending on the state. This may have an effect on the flow rate of other acyclic accesses to this device. More than one diagnostic messages may be present at a time. More than one diagnostic messages may be present particularly with device which provide a channel-granular diagnosis (1 diagnosis per IO point).

Output parameter "MoreDiagInfoAvailable" indicates that only a part of the present diagnostic messages has place in the entries provided.

Troubleshooting

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0001	0	Parameter error
INPUT_RANGE_ERROR	16#0001	1	IdentStr is not specified
INPUT_RANGE_ERROR	16#0001	2	DiagInfoAdr not set
INPUT_RANGE_ERROR	16#0001	3	SizoOfDiagInfo not set
INPUT_RANGE_ERROR	16#0001	4	SizoOfDiagInfo is not a multiple of PNIO_DIAGINFO
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
COMMUNICATION_ERROR	16#0106	0	Device “IdentStr” not configured
COMMUNICATION_ERROR	16#0003	*Profinet I/O controller diagnostics* • See General Error Codes , page 191, • See Special Error Codes, Profinet I/O Controller, page 193.	

Tab. 5-93: Error codes of the IL_PNIORemoteDeviceStateDetails function block

5.6 I&M (Identification and Maintenance) (IM Data)

5.6.1 IL_PNIO_IM0

Brief description

The structure type **IL_PNIO_IM0** serves as data classification of I&M0 (Read only).

Structural element	Data Type	Description
ManufacturerID	WORD	Manufacturer ID
OrderID	STRING(21)	Device part number
SerialNumber	STRING(17)	Device serial number
HardwareRevision	WORD	Hardware revision
SoftwareRevision	ARRAY[0..3] OF BYTE	Software revision
RevisionCounter	WORD	Revision counter
ProfileID	WORD	Profile number
ProfileSpecificType	WORD	Type of device

Structural element	Data Type	Description
IM_Version	ARRAY[0..1] OF BYTE	Version of the I&M profile
IM_Supported	WORD	Support of I&M data sets

Tab. 5-94: Structure of IL_PNIO_IM0

5.6.2 IL_PNIO_IM1

Brief description The structure type **IL_PNIO_IM1** serves as data classification of I&M1 (Read/ Write)

STRUCT	Data type	Description	Min. value	Max. value	Default value	Application
TagFunction	STRING(33)	Function of device	---	---	---	Rising edge at "Execute"
TagLocation	STRING(23)	Location of device	---	---	---	

Tab. 5-95: Structure of IL_PNIO_IM1

5.6.3 IL_PNIO_IM2

Brief description The structure type **IL_PNIO_IM2** serves as data classification of I&M2 (Read/ Write)

STRUCT	Data type	Description	Min. value	Max. value	Default value	Application
InstallationDate	STRING(17)	Installation data of the device	---	---	---	Rising edge at "Execute"
Reserved	ARRAY[0..37] OF BYTE	Reserved	---	---	---	

Tab. 5-96: Structure of IL_PNIO_IM2

5.6.4 IL_PNIO_IM3

Brief description The structure type **IL_PNIO_IM3** serves as data classification of I&M3 (Read/ Write)

STRUCT	Data type	Description	Min. value	Max. value	Default value	Application
Descriptor	STRING(55)	Comment of the device	---	---	---	Rising edge at "Execute"

Tab. 5-97: Structure of IL_PNIO_IM3

5.6.5 IL_PNIO_IM4

Brief description The structure type **IL_PNIO_IM4** serves as data classification of I&M4 (Read/ Write)

STRUCT	Data type	Description	Min. value	Max. value	Default value	Application
Signature	STRING(55)		---	---	---	Rising edge at "Execute"

Tab. 5-98: Structure of IL_PNIO_IM4

5.6.6 IL_PNIO_IM_RWMODE

Brief description The enumeration type **IL_PNIO_IM_RWMODE** serves to select the data set read / write mode for I&M0…I&M4.

Element	Value	Description
IL_PNIO_READ_IM0	16#0000	Read I&M0 data set
IL_PNIO_READ_IM1	16#0001	Read I&M1 data set
IL_PNIO_READ_IM2	16#0002	Read I&M2 data set
IL_PNIO_READ_IM3	16#0003	Read I&M3 data set
IL_PNIO_READ_IM4	16#0004	Read I&M4 data set
IL_PNIO_READ_IMALL	16#0005	Read I&M0...I&M4 data set
IL_PNIO_WRITE_IM1	16#0006	Write I&M1 data set
IL_PNIO_WRITE_IM2	16#0007	Write I&M2 data set
IL_PNIO_WRITE_IM3	16#0008	Write I&M3 data set
IL_PNIO_WRITE_IM4	16#0009	Write I&M4 data set
IL_PNIO_WRITE_IMALL	16#000A	Write I&M1...I&M4 data set

Tab. 5-99: Enumeration type IL_PNIO_IM_RWMODE

5.6.7 IL_PNIOReadWriteIMData

Brief description The function block **IL_PNIOReadWriteIMData** can read or write I&M data sets (identification & maintenance) of the Profinet I/O device using different read and write modes.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/IM Data
MTX	RIL_ProfinetIO/POUs/IM Data

Tab. 5-100: Reference table

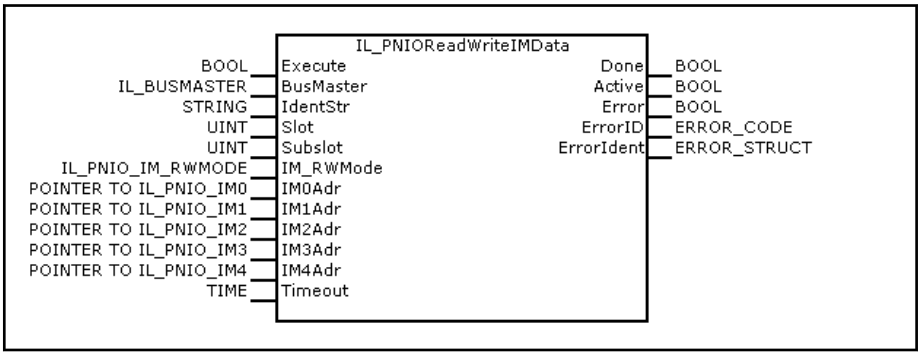


*Fig. 5-37: Interface description of the **IL_PNIOReadWriteIMData** function block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 184
	IdentStr	STRING	Identification string of the Profinet device.

	Name	Type	Comment
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example an I/O module)
	SubSlot	UINT	Addressing the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot).
	IM_RWMode	IL_PNIO_IM_RWMODE	Enumeration for the read/write mode of IM data Refer to IL_PNIO_IM_RWMODE, page 131
	IM0Adr	POINTER TO IL_PNIO_IM0	Address of the IM0 data structure for reading/writing Refer to IL_PNIO_IM0, page 130
	IM1Adr	POINTER TO IL_PNIO_IM1	Address of the IM1 data structure for reading/writing Refer to IL_PNIO_IM1, page 131
	IM2Adr	POINTER TO IL_PNIO_IM2	Address of the IM2 data structure for reading/writing Refer to IL_PNIO_IM2, page 131
	IM3Adr	POINTER TO IL_PNIO_IM3	Address of the IM3 data structure for reading/writing Refer to IL_PNIO_IM3, page 131
	IM4Adr	POINTER TO IL_PNIO_IM4	Address of the IM4 data structure for reading/writing Refer to IL_PNIO_IM4, page 131
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (ErrorTable PROFINETIO_TABLE)

Tab. 5-101: Function block IL_PNIOReadWriteIMData, I/O interface

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	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
IdentStr	STRING	---	---	---	Rising edge at "Execute"
Slot	UINT	0	16#7FFF	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8FFF	1	Rising edge at "Execute"
IM_RWMode	IL_PNIO_IM_RWMODE	IL_PNIO_READ_IM0	IL_PNIO_WRITE_IMALL	IL_PNIO_READ_IM0	Rising edge at "Execute"
IM0Adr	POINTER TO IL_PNIO_IM0	Not defined	Not defined	0	Not applicable

Name	Type	Min. value	Max. value	Default value	Application
IM1Adr	POINTER TO IL_PNIO_IM1	Not defined	Not defined	0	Rising edge at "Execute"
IM2Adr	POINTER TO IL_PNIO_IM2	Not defined	Not defined	0	Rising edge at "Execute"
IM3Adr	POINTER TO IL_PNIO_IM3	Not defined	Not defined	0	Rising edge at "Execute"
IM4Adr	POINTER TO IL_PNIO_IM4	Not defined	Not defined	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at "Execute"

Tab. 5-102: Min./max. and default values of the L_PNIOReadWriteIMData inputs

Functional description

The function block **IL_PNIOReadWriteIMData** can read or write I&M data sets (identification & maintenance) of the Profinet I/O device using different read and write modes. **IL_PNIOReadWriteIMData** accesses I&M functions of the memory index 0xAFF0 – 0xAFF4 with the following memory index attributes:

- I&M0 (required) read-only, index AFF0
- I&M1 (optional) read/write, index AFF1
- I&M2 (optional) read/write, index AFF2
- I&M3 (optional) read/write, index AFF3
- I&M4 (optional) read/write, index AFF4

The device is addressed via the Profinet device name in "IdentStr". It has to be part of the controller configuration. The data set on the Profinet device is addressed using the slot-subslot combination. Slot and subslot address the various logical or physical modules of a device.

Use "IM_RWMode" to select the operating mode of the function block. Thus, the function block can execute read/write operations for the memory index with data defined via "IM0Adr", "IM1Adr", "IM2Adr", "IM3Adr" und "IM4Adr".

- The mode **IL_PNIO_READ_IMALL** reads the IM0 data set first and then subsequently all other supported data sets (IM1...4) as per the “IM_Supported“ identified by IM0
- The mode **IL_PNIO_WRITE_IMALL** tries to subsequently write all the data sets IM1…4
- The function block tries to write or read the memory index in the modes **IL_PNIO_READ_IM1…4** / **IL_PNIO_WRITE_IM1…4** without checking whether the defined data set is supported

"IM_Supported" is a list of bits showing further available I&M data sets. Bit-0 : --; Bit-1: I&M1; Bit-2: I&M2; Bit-3: I&M3; Bit-4: I&M4;…. Bit-15: I&M15.

The "IM_Version" of the device has to be 1.x. Otherwise, the function block displays the error "I&M Version mismatch".

The function block displays the error “Response header is not matching” if the response of the requested I&M does not match the data in the following table.

BlockHeader	I&M0	I&M1	I&M2	I&M3	I&M4
Type	16#20	16#21	16#22	16#23	16#24

The rising edge at the "Execute" input starts the process. If the process was successful, the "Done" output is set to TRUE. If an error occurs, the "Error" bit is set and detailed error information is provided in "ErrorID" and "ErrorIdent".

Example program

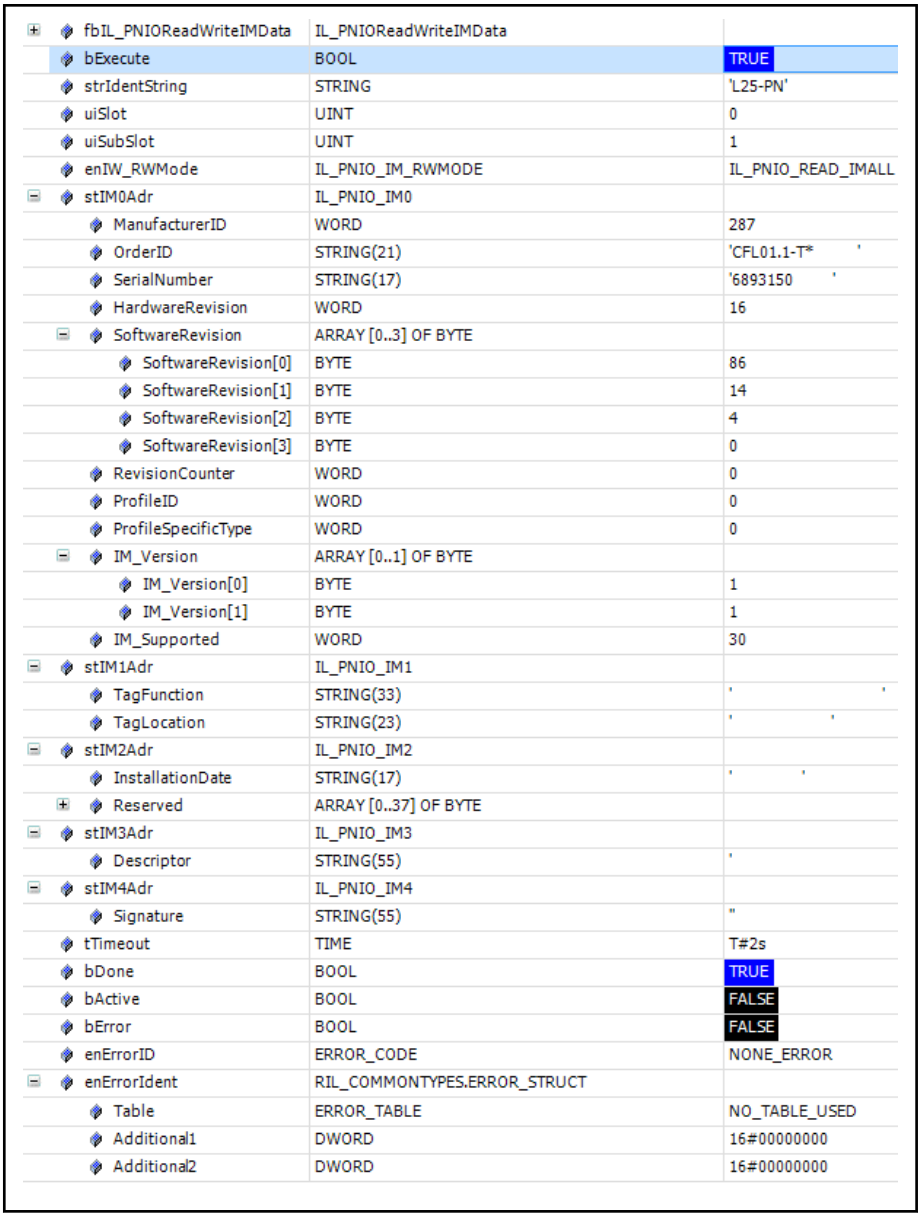
This is an example project to read I&M details of 0..4 by executing the slave device once. The Profinet slave device is at slot 0. Its station name is "PNIO-device-1". The rising edge at the "bExecute" input executes the process and reads I&M0 data first and then the supported I&Ms.

Declaration

```
PROGRAM ProgIL_PNIOReadWriteIMData
VAR
    fbIL_PNIOReadWriteIMData: IL_PNIOReadWriteIMData; // FB Instance
    bExecute:          BOOL; //Function block is running when TRUE
    strIdentString:          STRING := 'PNIO-device-1'; //Station name of PN BK
    uiSlot:              UINT := 16#00; //Addressing of the module slot.
                                //The slot addresses a logical or a physical module
    uiSubSlot:          UINT := 16#01; //Addressing of the submodule.
    enIW_RWMode:     IL_PNIO_IM_RWMODE := IL_PNIO_IM_RWMODE.IL_PNIO_READ_IMALL;
    stIM0Adr:         IL_PNIO_IM0; //Address of read/write IM0 data structure.
    stIM1Adr:         IL_PNIO_IM1; //Address of read/write IM1 data structure.
    stIM2Adr:         IL_PNIO_IM2; //Address of read/write IM2 data structure.
    stIM3Adr:         IL_PNIO_IM3; //Address of read/write IM3 data structure.
    stIM4Adr:         IL_PNIO_IM4; //Address of read/write IM4 data structure.
    tTimeout:          TIME := T#2S; //Timeout monitoring of the function block
    bDone:              BOOL; //Function Block is completed its task
    bActive:          BOOL; //Function Block is working on its actual task
    bError:              BOOL; //Indicates an error
    enErrorID:          RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent:          RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
END_VAR
```

Implementation

```
fbIL_PNIOReadWriteIMData(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    Slot:= uiSlot,
    SubSlot:= uiSubSlot,
    IM_RWMode:= enIW_RWMode,
    IM0Adr:= ADR(stIM0Adr),
    IM1Adr:= ADR(stIM1Adr),
    IM2Adr:= ADR(stIM2Adr),
    IM3Adr:= ADR(stIM3Adr),
    IM4Adr:= ADR(stIM4Adr),
    Timeout:= tTimeout);
```



fbIL_PNIOReadWriteIMData	IL_PNIOReadWriteIMData	
bExecute	BOOL	TRUE
strIdentString	STRING	'L25-PN'
uiSlot	UINT	0
uiSubSlot	UINT	1
enIW_RWMode	IL_PNIO_IM_RWMODE	IL_PNIO_READ_IMALL
stIM0Adr	IL_PNIO_IM0	
ManufacturerID	WORD	287
OrderID	STRING(21)	'CFL01.1-T* '
SerialNumber	STRING(17)	'6893150 '
HardwareRevision	WORD	16
SoftwareRevision	ARRAY [0..3] OF BYTE	
SoftwareRevision[0]	BYTE	86
SoftwareRevision[1]	BYTE	14
SoftwareRevision[2]	BYTE	4
SoftwareRevision[3]	BYTE	0
RevisionCounter	WORD	0
ProfileID	WORD	0
ProfileSpecificType	WORD	0
IM_Version	ARRAY [0..1] OF BYTE	
IM_Version[0]	BYTE	1
IM_Version[1]	BYTE	1
IM_Supported	WORD	30
stIM1Adr	IL_PNIO_IM1	
TagFunction	STRING(33)	' '
TagLocation	STRING(23)	' '
stIM2Adr	IL_PNIO_IM2	
InstallationDate	STRING(17)	' '
Reserved	ARRAY [0..37] OF BYTE	
stIM3Adr	IL_PNIO_IM3	
Descriptor	STRING(55)	'
stIM4Adr	IL_PNIO_IM4	
Signature	STRING(55)	"
tTimeout	TIME	T#2s
bDone	BOOL	TRUE
bActive	BOOL	FALSE
bError	BOOL	FALSE
enErrorID	ERROR_CODE	NONE_ERROR
enErrorIdent	RIL_COMMONTYPES.ERROR_STRUCT	
Table	ERROR_TABLE	NO_TABLE_USED
Additional1	DWORD	16#00000000
Additional2	DWORD	16#00000000

Fig. 5-38: Executing the function block online

Error handling

The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_INVALID_ERROR	16#0001	16#0000	IMxAdr is not set
INPUT_INVALID_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_INVALID_ERROR	16#0001	16#0004	IM_RWMode is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	

ErrorID	Additional1	Additional2	Remarks
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header not matching
DEVICE_ERROR	16#0007	16#0000	I&M Version mismatch, device I&M Major RevisionID must be 1
SYSTEM_ERROR	16#0101	Internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	16#0000 16#0001 16#0002	Statemachine error

Tab. 5-103: Error codes IL_PNIOReadWriteIMData

5.7 Communication interface (IO-Link)

5.7.1 IL_IOL_DEVICE_STATUS

Brief description The IL_IOL_DEVICE_STATUS enumeration is used to give information about the Device condition (diagnosis).

Element	Value	Description
STATUS_INVALID	16#FFFF	Status is unavailable.
WORKING_PROPERLY	16#0000	Device is operating properly.
MAINTENANCE_REQUIRED	16#0001	Although the Process Data are valid, internal diagnostics indicate that the Device is close to loose its ability of correct functioning.
OUT_OF_SPECIFICATION	16#0002	Although the Process Data are valid, internal diagnostics indicate that the Device is operating outside its specified measuring range or environmental conditions.
FUNCTIONAL_CHECK	16#0003	Process Data temporarily invalid due to intended manipulations on the Device.
FAILURE	16#0004	Process Data invalid due to malfunction in the Device or its peripherals.
RESERVED	16#00FF	Reserved.

Tab. 5-104: Enumeration type IL_IOL_DEVICE_STATUS

5.7.2 IL_IOL_DEVICE_STATUS_DETAILS

Brief description This structure is used to display the detailed device status of IO-Link devices.

Structural element	Value	Description
EventQualifier	IL_IOL_EVENT_QUALIFIER	Detailed information of event. IL_IOL_EVENT_QUALIFIER, page 137
EventCode	WORD	The EventCode entry contains the identifier of an actual Event.

Tab. 5-105: Structure of IL_IOL_DEVICE_STATUS_DETAILS

5.7.3 IL_IOL_EVENT_QUALIFIER

Brief description This structure is used to display the detailed event information of IO-Link devices.

Structural element	Data type	Description
Instance	IL_IOL_EVENT_QUALIFI-ER_INSTANCE	Indicates the particular source (instance) of an Event IL_IOL_EVENT_QUALIFIER_INSTANCE, page 138
Source	IL_IOL_EVENT_QUALIFI-ER_SOURCE	Indicates the source of the Event IL_IOL_EVENT_QUALIFIER_SOURCE, page 138
QualifierType	IL_IOL_EVENT_QUALIFI-ER_TYPE	Indicates the Event category IL_IOL_EVENT_QUALIFIER_TYPE, page 138
Mode	IL_IOL_EVENT_QUALIFI-ER_MODE	Indicates the Event mode IL_IOL_EVENT_QUALIFIER_MODE, page 138

Tab. 5-106: Structure of IL_IOL_EVENT_QUALIFIER

5.7.4 IL_IOL_EVENT_QUALIFIER_INSTANCE

Brief description The IL_IOL_EVENT_QUALIFIER_INSTANCE enumeration is used to indicates the particular source (instance) of an Event.

Element	Value	Description
UNKNOWN	16#0000	Unknown
RESERVED	16#0001	Reserved
APPLICATION	16#0004	Application

Tab. 5-107: Enumeration type IL_IOL_EVENT_QUALIFIER_INSTANCE

5.7.5 IL_IOL_EVENT_QUALIFIER_MODE

Brief description The IL_IOL_EVENT_QUALIFIER_MODE enumeration is used to indicates the Event mode.

Element	Value	Description
RESERVED	016#000	Reserved
SINGLE_SHOT	016#001	Event single shot
DISAPPEARS	016#002	Event disappears
APPEARS	016#003	Event appears

Tab. 5-108: Enumeration type IL_IOL_EVENT_QUALIFIER_MODE

5.7.6 IL_IOL_EVENT_QUALIFIER_SOURCE

Brief description The IL_IOL_EVENT_QUALIFIER_SOURCE enumeration is used to indicates the source of the Event.

Element	Value	Description
DEVICE	16#0000	Device is source. (Remote)
MASTER	16#0001	Master is source. (Local)

Tab. 5-109: Enumeration type IL_IOL_EVENT_QUALIFIER_SOURCE

5.7.7 IL_IOL_EVENT_QUALIFIER_TYPE

Brief description The IL_IOL_EVENT_QUALIFIER_TYPE enumeration is used to indicates the Event category.

Element	Value	Description
RESERVED	016#000	Reserved
NOTIFICATION	016#001	Notification
WARNING	016#002	Warning
ERROR	016#003	Error

Tab. 5-110: Enumeration type IL_IOL_EVENT_QUALIFIER_TYPE

5.7.8 IL_IOL_SERVICE

Brief description The IL_IOL_SERVICE enumeration is used to select the IOL services.

Element	Value	Description
CANCEL	16#0000	Cancel / Release IOL_CALL
IDLE	16#0001	Idle sequence
WRITE	16#0002	Write data
READ	16#0003	Read data

Tab. 5-111: Enumeration type IL_IOL_SERVICE

5.7.9 IL_IOL_BACKUP_MODE

Brief description The enumeration type **IL_IOL_BACKUP_MODE** selects the mode of reading and storing parameters from the IOL device to the controller.

ENUM	Constant	Description
NO_BLOCK	16#0000	Non-Block mode, single parameter is read in this mode
BLOCK	16#0001	Block mode, Multiple parameters are read in this mode by blocking write requests from other sources to ensure data integrity

Tab. 5-112: Enumeration type IL_IOL_BACKUP_MODE

5.7.10 IL_IOL_RESTORE_MODE

Brief description The enumeration type **IL_IOL_RESTORE_MODE** is used to select the mode to restore parameters from the controller to the IOL device.

ENUM	Constant	Description
BLOCK_CHECK	16#00	Download of parameters with parameter blocking and with validity checking
NO_BLOCK_CHECK	16#01	Download of parameters without parameter blocking and online validity checking
NO_BLOCK_NO_CHECK	16#02	Download of parameters without parameter blocking and with validity checking

Tab. 5-113: Enumeration type IL_IOL_RESTORE_MODE

5.7.11 IL_PNIOIoLinkCall

Brief description The function block **IL_PNIOIoLinkCall** is used to handle the read and write services of IOL data objects via Profinet-IO-Link-Gateways.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-114: Reference table

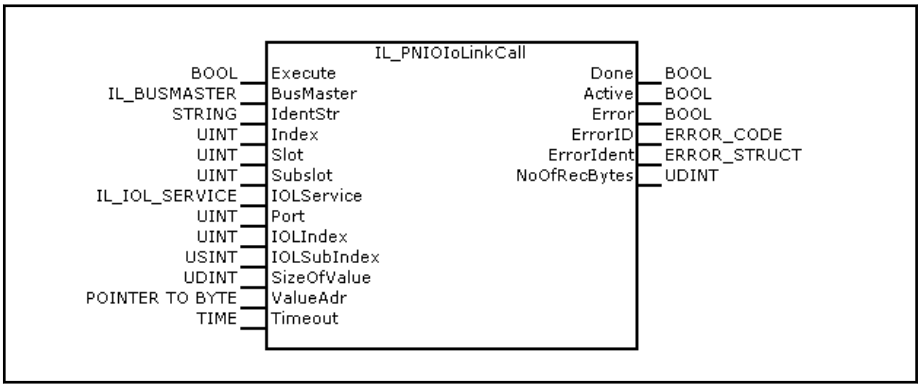


*Fig. 5-39: Interface Description of the **IL_PNIOIoLinkCall** Function Block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the PROFINET device.
	Index	UINT	255 – Regular index Call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example, an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	IOLService	IL_IOL_SERVICE	Enumeration for IOL service. IL_IOL_SERVICE, page 139
	Port	UINT	1..63: IO-Link Port Number
	IOLIndex	UINT	0..32767 - IO Link Index, 65535 – Port function
	IOLSubIndex	USINT	0..255: IO Link Subindex
	SizeOfValue	UDINT	Size of ValueAdr array. The length of the Write (max. 232 Bytes) and Read data
	ValueAdr	POINTER TO BYTE	Initial address of the request or response data array.
	Timeout	TIME	Timeout monitoring of the function block

	Name	Type	Comment
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnosis in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable PROFINETIO_TABLE)
	NoOfRecBytes	UDINT	Number of bytes received as response

Tab. 5-115: FB IL_PNIOIoLinkCall I/O interface

Min./Max. And Default Values: The following table describes the minimum, maximum and default values of the function block inputs.

Name	Type	Min. value	Max. Value	Default Value	Application
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING			""	Rising edge at "Execute"
Index	UINT			255	Rising edge at "Execute"
Slot	UINT	0	32767	0	Rising edge at "Execute"
SubSlot	UINT	1	36863	1	Rising edge at "Execute"
IOLService	IL_IOL_SERVICE			IDLE	Rising edge at "Execute"
Port	UINT	0	255	0	Rising edge at "Execute"
IOLIndex	UINT			0	Rising edge at "Execute"
IOLSubIndex	USINT			0	Rising edge at "Execute"
SizeOfValue	UDINT	0	Write – 232 Bytes	0	Rising edge at "Execute"
ValueAdr	POINTER TO BYTE			0	Rising edge at "Execute"
Timeout	TIME	T#0ms		T#2s	Rising edge at "Execute"

Tab. 5-116: Minimum / maximum values and default values of the IL_PNIOIo-LinkCall inputs

Functional description The function block **IL_PNIOIoLinkCall** is used to handle the read and write services of IOL data objects via Profinet-IO-Link-Gateways.

The read and write service depends on the selection made at IOLService input. The IOL Data objects can be directly read and write by using IOLIndex and IOLSubIndex.

The PNIO IOL call PDU contains PN-IO Header, Call- Header, IOL-Header and Body.

Coding of IOL Call request PDU is defined in the table below.

Section	Content	Size	Coding	Definitions
DP-V1-Header	Function_Num	1 Octet	5FH	Fix ("Write Record")
	Slot_Number	1 Octet	x	Slot: IOL-M
	Index	1 Octet	255 0...254	Regular index (CALL) Index redirection (cap)
	Length	1 Octet	0...241	Length of data including header
Call -Header	Extended_Function_Num	1 Octet	08h	Indicates "CALL", fix
	Entity_Port	1 Octet	0 1...63 64...255	IOL-M Port number Reserved
	FI_Index	2 Octets	65098	IOL-Header is following
IOL-Header	Control	1 Octet	0...255	"Coding of Control" (9.3.3)
	IOL_Index	2 Octet	0...32767 65535	IOL-D data: Index Port function invocation
	IOL_Subindex	1 Octet	0...255	IOL-D data: Subindex or Port function
Body	IOL_Data_Object	232 Octets maximum		"Write" Data
NOTE	The coding of the DP-V1-Header within this table is exemplary for PB-DP. The PN-IO-Header is characterized by API /Slot /Subslot /Index.			

Fig. 5-40: Coding of an IOL_CALL request PDU

The user should provide all the inputs for the function block with respect to IOL call PDU. PN-IO header, Extended_Fuction_Num & FI_Index are handled internally. During write request data to be given in the Array defined at "ValueAdr" and number of data to be written at "SizeOfValue" input. During read request user should ensure the "SizeOfValue" is in line with response data. For both read and write response operation header is checked to validate successful communication. In read operation the response data is copied to "ValueAdr" and total number of received data bytes will be updated in "NoOfRecBytes".

The rising edge at "Execute" input process the operation and after successful completion sets done to true.

If any error then the error bit is set and the detailed error information is available at "ErrorID" and "ErrorIdent".

Handling Specials:

Parameters/Values requests to values greater than 16 Bit and Datatype e.g. UINT have to be swapped.

Example Program

1. In this example project "vender name" is read from the device connected to port 0.

   The profinet bus coupler is available in slot 0.

   The data object vender name is assigned to the IOL index 16#10.

   The rising edge at "bExecute" input executes the operation and data read from the device is available at "arValue".

   The length of the vender name is displayed at "udiNoOfRecvBytes".

Declaration

```
PROGRAM prgIL_PNIOIoLinkCall
VAR
```

```
    fbIL_PNIOIoLinkCall: IL_PNIOIoLinkCall;
    bExecute: BOOL := FALSE;
    uiIndex: UINT := 16#FF; //Index to read IOL data
    strIdentString: STRING := 'iolballuff'; //Station name of PN BK
    uiSlot: UINT := 16#01;
    uiSubSlot: UINT := 16#01;
    enIOLServiceRead: IL_IOL_SERVICE := IL_IOL_SERVICE.READ;
    uiPort: UINT := 16#01;
    uiIOLIndex: UINT := 16#10;
    usiIOLSubIndex: USINT := 16#00;
    arValue: ARRAY[1..232] OF BYTE;
    tTimeout: TIME := T#1S;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: RIL_COMMONTYPES.ERROR_CODE;
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT;
    udiNoOfRecvBytes: UDINT;
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkCall(
    Execute:= bExecute,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    Index:= uiIndex,
    Slot:= uiSlot,
    Subslot:= uiSubSlot,
    IOLService:= enIOLServiceRead,
    Port:= uiPort,
    IOLIndex:= uiIOLIndex,
    IOLSubIndex:= usiIOLSubIndex,
    SizeOfValue:= SIZEOF(arValue),
    ValueAdr:= ADR(arValue),
    Timeout:= tTimeout,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    NoOfRecBytes=> udiNoOfRecvBytes);
```

2. In this example data is written to the index 16#40 with value 10 and 11.

Declaration

```
PROGRAM prgIL_PNIOIoLinkCall
VAR
    fbIL_PNIOIoLinkCall: IL_PNIOIoLinkCall;
    bExecute: BOOL := FALSE;
    uiIndex: UINT := 16#FF; //Index to read IOL data
    strIdentString: STRING := 'iolballuff'; //Station name of PN BK
    uiSlot: UINT := 16#01;
    uiSubSlot: UINT := 16#01;
    enIOLServiceWrite: IL_IOL_SERVICE := IL_IOL_SERVICE.WRITE;
    uiPort: UINT := 16#01;
    uiIOLIndex: UINT := 16#10;
    usiIOLSubIndex: USINT := 16#00;
udiSizeOfWriteData: UDINT := 16#02;
    arValue: ARRAY[1..232] OF BYTE;
    tTimeout: TIME := T#1S;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: RIL_COMMONTYPES.ERROR_CODE;
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT;
    udiNoOfRecvBytes: UDINT;
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkCall(
    Execute:= bExecute,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    Index:= uiIndex,
    Slot:= uiSlot,
    Subslot:= uiSubSlot,
```

```
IOLService:= enIOLServiceWrite,
Port:= uiPort,
IOLIndex:= uiIOLIndex,
IOLSubIndex:= usiIOLSubIndex,
SizeOfValue:=udiSizeOfWriteData,
ValueAdr:= ADR(arValue),
Timeout:= tTimeout,
Done=> bDone,
Active=> bActive,
Error=> bError,
ErrorID=> enErrorID,
ErrorIdent=> enErrorIdent,
NoOfRecBytes=> udiNoOfRecvBytes);
```



Before using the example project check whether device has same index and values compatible for that.

Troubleshooting

The function block uses error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	ValueAdr is not set
INPUT_RANGE_ERROR	16#0001	16#0001	SizeOfValue is not set. SizeOfValue > 232 for Write request
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_RANGE_ERROR	16#0001	16#0003	IOLService is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header is not matching
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	internal use	State machine error
Special Error Codes, IO Link Master (IOL-M errors)			
COMMUNICATION_ERROR	16#3000	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request
COMMUNICATION_ERROR	16#3000	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#3000	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#3000	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#3000	16#8001	Wrong index. IOL_Index 32767 ... 65535
COMMUNICATION_ERROR	16#3000	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#3000	16#8003	Wrong port function. Port function not supported
Special Error Codes, IO Link Device (application related errors)			

ErrorID	Additional1	Additional2	Remarks
COMMUNICATION_ERROR	16#3001	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#3001	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#3001	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#3001	16#8023	Access denied
COMMUNICATION_ERROR	16#3001	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#3001	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#3001	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#3001	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#3001	16#8034	Parameter length under run
COMMUNICATION_ERROR	16#3001	16#8035	Function not available
COMMUNICATION_ERROR	16#3001	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#3001	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#3001	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#3001	16#8082	Application not ready
COMMUNICATION_ERROR	16#3001	16#8001..16#81FF	Vendor specific

Tab. 5-117: Error codes PNIO_IO-Link

5.7.12 IL_PNIOIoLinkGetDeviceAccessLock

Brief description The function block **IL_PNIOIoLinkGetDeviceAccessLock** reads out the status of IO-Link device parameter Device Access Locks of IO-Link device connected on Profinet coupler. This parameter contains the following lock categories:

- Parameter write access (optional)
- Data Storage (mandatory if device supports data storage)
- Local parameterization (option)
- Local user interface operation (option)

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-118: Reference table

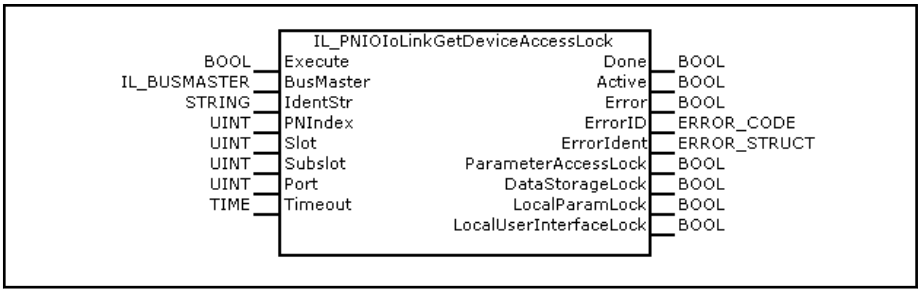


*Fig. 5-41: Interface Description of the **IL_PNIOIoLinkGetDeviceAccessLock** Function Block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the PROFINET device.
	PNIndex	UINT	255 – Regular index Call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example, an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	Port	UINT	1..63: IO-Link Port Number
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnosis in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable PROFINETIO_TABLE)
	ParameterAccess-Lock	BOOL	Parameter write access lock status FALSE: Unlocked, TRUE: Locked
	DataStorageLock	BOOL	Data storage lock status FALSE: Unlocked, TRUE: Locked
	LocalParamLock	BOOL	Local parameterisation lock status FALSE: Unlocked, TRUE: Locked
	LocalUserInterface-Lock	BOOL	Local user interface lock status FALSE: Unlocked, TRUE: Locked

Tab. 5-119: FB IL_PNIOIoLinkGetDeviceAccessLock I/O interface

Min./Max. And Default Values: The following table describes the minimum, maximum and default values of the function block inputs.

Name	Type	Min. value	Max. Value	Default Value	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING	---	---	---	Rising edge at "Execute"
PNIndex	UINT	0	16#ffff	0	Rising edge at "Execute"
Slot	UINT	0	16#7fff	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8fff	1	Rising edge at "Execute"
Port	UINT	0	255	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	----	T#2s	Rising edge at "Execute"

Tab. 5-120: Minimum / maximum values and default values of the IL_PNIOIo-LinkGetDeviceAccessLock inputs

Functional description

The function block **IL_PNIOIoLinkGetDeviceAccessLock** reads out the status of IO-Link device parameter Device Access Locks connected on desired IO-Link port. This Function block reads the IO-Link index 16#000C. This parameter contains the following lock categories and support of these categories is device specific:

- Parameter write access (optional)
- Data Storage (mandatory if device supports data storage)
- Local parameterization (option)
- Local user interface operation (option)

If any category of the parameter is not supported by connected device, the status of that parameter will be displayed as FALSE by default. The following table specifies the device locking possibilities.

The device is addressed via the PROFINET device name in "IdentStr", port number in "Port". It must be part of the configuration of the controller. The data set on the PROFINET device is addressed via the Slot-SubSlot combination as well as via the PNIndex.

The rising edge at Execute input starts the operation and after successful completion the output Done is set to TRUE. If any error then the Error bit is set and the detailed error information is available at ErrorID and ErrorIdent.

Device locking possibilities

Bit	Category	Definition
0	Parameter write access (optional)	0: unlocked (default) 1: locked
1	Data Storage (mandatory if device supports data storage)	0: unlocked (default) 1: locked (see Note)
2	Local parameterization (option)	0: unlocked (default) 1: locked
3	Local user interface operation (option)	0: unlocked (default) 1: locked

Bit	Category	Definition
4-15	Reserved	
Note: The Master reads the parameter State_Properties/ State of Data Storage prior to any action		

Example Program

This is the example project to read, status of Device Access Locks from the IO-Link device connected on port 5.

The profinet bus coupler is available in slot 1 with "iolballuff" as station name.

The rising edge at bExecute input executes the operation and reads the data of "ParameterAccessLock", "DataStorageLock", "LocalParamLock" and "LocalUserInterfaceLock" from the device.

Declaration

```
PROGRAM ProgIL_PNIOIoLinkGetDeviceAccessLock
VAR
    fb_PNIOIoLinkGetDeviceAccessLock: IL_PNIOIoLinkGetDeviceAccessLock;
    bExecute: BOOL; //Function block is running when TRUE
    uiIndex: UINT := 16#FF; //Index to read IOL data
    strIdentString: STRING := 'iolballuff'; //Station name of PN BK
    uiSlot: UINT := 16#01; //Addressing of the module slot.
                           //The slot addresses a logical or a physical module
    uiSubSlot: UINT := 16#01; //Addressing of the submodule.
    uiPort: UINT := 16#06; //IO-Link Port Number
    tTimeout: TIME := T#2S; //Timeout monitoring of the function block
    bDone: BOOL; //Function Block is completed its task
    bActive: BOOL; //Function Block is working on its actual task
    bError: BOOL; //Indicates an error
    enErrorID: RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
    bParameterAccessLock: BOOL; //Parameter write access lock status
    bDataStorageLock: BOOL; //Data storage lock status
    bLocalParamLock: BOOL; //Local parameterisation lock status
    bLocalUserInterfaceLock: BOOL; //Local user interface lock status
END_VAR
```

Implementation

```
fb_PNIOIoLinkGetDeviceAccessLock(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    PNIndex:= uiIndex,
    Slot:= uiSlot,
    Subslot:= uiSubSlot,
    Port:= uiPort,
    Timeout:= tTimeout,
    ParameterAccessLock=> bParameterAccessLock,
    DataStorageLock=> bDataStorageLock,
    LocalParamLock=> bLocalParamLock,
    LocalUserInterfaceLock=> bLocalUserInterfaceLock);
```

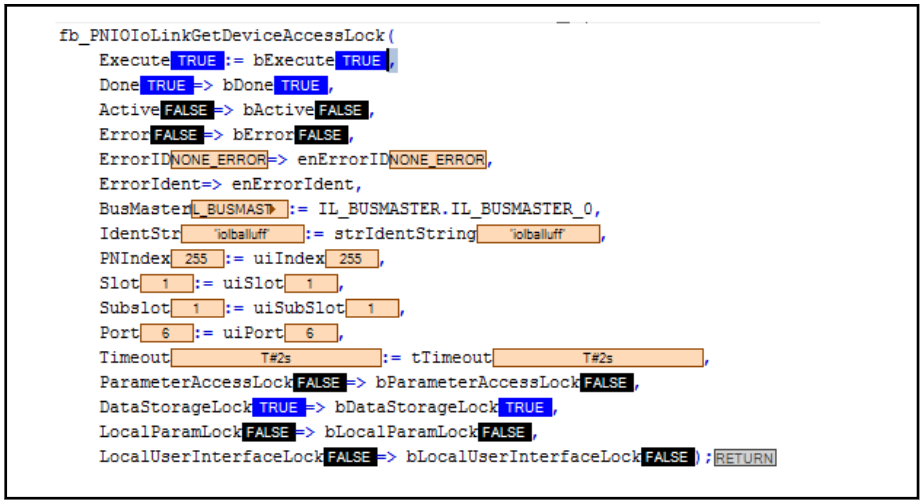


Fig. 5-42: Online execution of FB

Troubleshooting

The function block uses error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	ValueAdr is not set
INPUT_RANGE_ERROR	16#0001	16#0001	SizeOfValue is not set. SizeOfValue > 232 for Write request
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_RANGE_ERROR	16#0001	16#0003	IOLService is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header is not matching
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	internal use	State machine error
Special Error Codes, IO Link Master (IOL-M errors)			
COMMUNICATION_ERROR	16#3000	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request
COMMUNICATION_ERROR	16#3000	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#3000	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#3000	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#3000	16#8001	Wrong index. IOL_Index 32767 ... 65535

ErrorID	Additional1	Additional2	Remarks
COMMUNICATION_ERROR	16#3000	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#3000	16#8003	Wrong port function. Port function not supported
Special Error Codes, IO Link Device (application related errors)			
COMMUNICATION_ERROR	16#3001	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#3001	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#3001	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#3001	16#8023	Access denied
COMMUNICATION_ERROR	16#3001	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#3001	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#3001	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#3001	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#3001	16#8034	Parameter length under run
COMMUNICATION_ERROR	16#3001	16#8035	Function not available
COMMUNICATION_ERROR	16#3001	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#3001	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#3001	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#3001	16#8082	Application not ready
COMMUNICATION_ERROR	16#3001	16#8001..16#81FF	Vendor specific

Tab. 5-121: Error codes PNIO_IO-Link

5.7.13 IL_PNIOIoLinkIdent

Brief description The function block **IL_PNIOIoLinkIdent** is used to read out basic information to identify an IO-Link device connected to a desired IO-Link port on profinet.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-122: Reference table

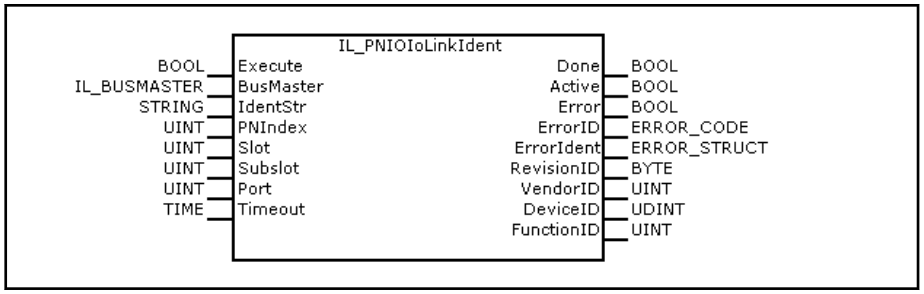


*Fig. 5-43: Interface Description of the **IL_PNIOIoLinkIdent** Function Block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the PROFINET device.
	PNIndex	UINT	255 – Regular index Call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example, an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	Port	UINT	1..63: IO-Link Port Number
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnosis in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable PROFINETIO_TABLE)
	RevisionID	BYTE	ID of the used protocol version for implementation. (XXXX YYYY of byte) the ‘X’ gives MajorRevisionID and ‘Y’ gives MinorRevisionId.
	VendorID	UINT	Unique vendor identification.
	DeviceID	UDINT	Unique Device identification allocated by a vendor.
	FunctionID	UINT	Reserved (future use).

Tab. 5-123: FB IL_PNIOIoLinkIdent I/O interface

Min./Max. And Default Values: The following table describes the minimum, maximum and default values of the function block inputs.

Name	Type	Min. value	Max. Value	Default Value	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING	---		‘’	Rising edge at "Execute"
PNIndex	UINT	0	16#ffff	255	Rising edge at "Execute"
Slot	UINT	0	16#7fff	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8fff	1	Rising edge at "Execute"
Port	UINT	0	255	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	----	T#5s	Rising edge at "Execute"

Tab. 5-124: Minimum / maximum values and default values of the IL_PNIOIoLinkIdent inputs

Functional description

The function block **IL_PNIOIoLinkIdent** is used to read out basic information to identify an IO-Link device connected to a desired IO-Link port.

The device is addressed via the PROFINET device name in "IdentStr". It must be part of the configuration of the controller. The data set on the PROFINET device is addressed via the Slot-SubSlot combination as well as via the PNIndex.

The function block reads the direct parameters of page 1 of an IOL device. Page 1 range from 0x00 to 0x0F. It comprises of communication protocol and identification parameters. The function block reads only identification parameters of page 1 and its subIndex range from 0x04 to 0x0D.

Example Program

This example project is configured to read the identification parameters of device BNI IOL-801-000-Z036 connected at port 6 of profinet gateway. The rising edge at "bExecute" input executes the read operation from the device and identification data is displayed at output attributes.

Declaration

```
PROGRAM ProgIL_PNIOIoLinkIdent
VAR
    fbIL_PNIOIoLinkIdent: IL_PNIOIoLinkIdent;
    bExecute:         BOOL; //Function block is running when TRUE
    uiIndex:         UINT := 16#FF; //Index to read IOL data
    strIdentString: STRING := 'iolballuff'; //Station name of PN BK
    uiSlot:         UINT := 16#01; //Addressing of the module slot.
                         //The slot addresses a logical or a physical module
    uiSubSlot:         UINT := 16#01; //Addressing of the submodule.
    uiPort:         UINT := 16#06; //IO-Link Port Number
    tTimeout:         TIME := T#2S; //Timeout monitoring of the function block
    bDone:             BOOL; //Function Block is completed its task
    bActive:         BOOL; //Function Block is working on its actual task
    bError:         BOOL; //Indicates an error
    enErrorID:         RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent:     RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
    byRevisionID:     BYTE; // ID of the used protocol version for implementation.
    //(XXXX YYYY of byte) the ‘X’ gives RevisionIDMajor and ‘Y’ gives RevisionIdMinor.
    uiVendorID:     UINT; // Unique vendor identification.
    udiDeviceID:     UDINT;    //Unique Device identification allocated BY a vendor.
```

```
    uiFunctionID:     UINT;    //Reserved (Engineering shall set both octets to "16#00")
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkIdent(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    PNIndex:= uiIndex,
    Slot:= uiSlot,
    Subslot:= uiSubSlot,
    Port:= uiPort,
    Timeout:= tTimeout,
    RevisionID=> byRevisionID,
    VendorID=> uiVendorID,
    DeviceID=> udiDeviceID,
    FunctionID=> uiFunctionID);
```

Expression	Type	Value
fbIL_PNIOIoLinkIdent	IL_PNIOIoLinkIdent	
bExecute	BOOL	TRUE
uiIndex	UINT	16#00FF
strIdentString	STRING	'iolbalduff'
uiSlot	UINT	16#0001
uiSubSlot	UINT	16#0001
uiPort	UINT	16#0006
tTimeout	TIME	T#2s
bDone	BOOL	TRUE
bActive	BOOL	FALSE
bError	BOOL	FALSE
enErrorID	ERROR_CODE	NONE_ERROR
enErrorIdent	RIL_COMMONTYPES.ERROR_STRUCT	
byRevisionID	BYTE	16#11
uiVendorID	UINT	16#0378
udiDeviceID	UDINT	16#00050A02
uiFunctionID	UINT	16#0000

Fig. 5-44: Online execution of FB

Troubleshooting

The function block uses error table **PROFINETIO_TABLE**(ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	ValueAdr is not set
INPUT_RANGE_ERROR	16#0001	16#0001	SizeOfValue is not set. SizeOfValue > 232 for Write request
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_RANGE_ERROR	16#0001	16#0003	IOLService is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	

ErrorID	Additional1	Additional2	Remarks
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header is not matching
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	internal use	State machine error
Special Error Codes, IO Link Master (IOL-M errors)			
COMMUNICATION_ERROR	16#3000	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request
COMMUNICATION_ERROR	16#3000	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#3000	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#3000	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#3000	16#8001	Wrong index. IOL_Index 32767 ... 65535
COMMUNICATION_ERROR	16#3000	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#3000	16#8003	Wrong port function. Port function not supported
Special Error Codes, IO Link Device (application related errors)			
COMMUNICATION_ERROR	16#3001	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#3001	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#3001	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#3001	16#8023	Access denied
COMMUNICATION_ERROR	16#3001	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#3001	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#3001	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#3001	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#3001	16#8034	Parameter length under run
COMMUNICATION_ERROR	16#3001	16#8035	Function not available
COMMUNICATION_ERROR	16#3001	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#3001	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#3001	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#3001	16#8082	Application not ready
COMMUNICATION_ERROR	16#3001	16#8001..16#81FF	Vendor specific

Tab. 5-125: Error codes PNIO_IO-Link

5.7.14 IL_PNIOIoLinkIdentDetails

Brief description The function block **IL_PNIOIoLinkIdentDetails** is used to read identification parameters and device specific information of an IO-Link device connected to a desired IO-Link port on profinet.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-126: Reference table

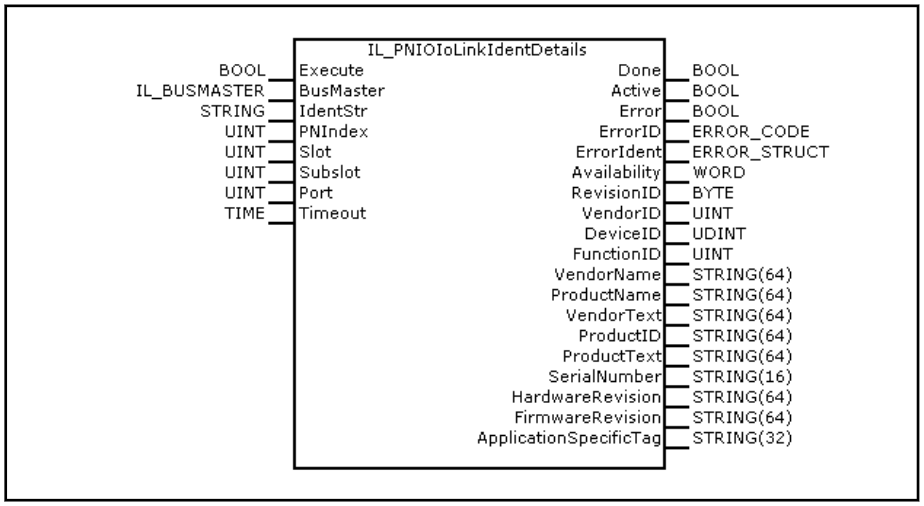


*Fig. 5-45: Interface Description of the **IL_PNIOIoLinkIdentDetails** Function Block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the PROFINET device.
	PNIndex	UINT	255 – Regular index Call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example, an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	Port	UINT	1..63: IO-Link Port Number
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid

	Name	Type	Comment
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnosis in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable PROFINETIO_TABLE)
	RevisionID	BYTE	ID of the used protocol version for implementation. (XXXX YYYY of byte) the ‘X’ gives MajorRevisionID and ‘Y’ gives MinorRevisionId
	VendorID	UINT	Unique vendor identification
	DeviceID	UDINT	Unique Device identification allocated by a vendor
	FunctionID	UINT	Reserved (future use)
	VendorName	STRING(64)	Name of the Vendor
	ProductName	STRING(64)	Name of the Product
	VendorText	STRING(64)	Vendor text
	ProductID	STRING(64)	Unique Product identification allocated by a vendor
	ProductText	STRING(64)	Product Text
	SerialNumber	STRING(16)	Vendor specific serial number
	HardwareRevision	STRING(64)	Hardware revision of the Device
	FirmwareRevision	STRING(64)	Firmware revision of the Device
	ApplicationSpecific-Tag	STRING(32)	It can serve as a "tag function" (role of the Device) or a "tag location" (location of the Device)

Tab. 5-127: FB IL_PNIOIoLinkIdentDetails I/O interface

Min./Max. And Default Values: The following table describes the minimum, maximum and default values of the function block inputs.

Name	Type	Min. value	Max. Value	Default Value	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING	---		‘’	Rising edge at "Execute"
PNIndex	UINT	0	16#ffff	255	Rising edge at "Execute"
Slot	UINT	0	16#7fff	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8fff	1	Rising edge at "Execute"

Name	Type	Min. value	Max. Value	Default Value	Application
Port	UINT	0	255	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	----	T#5s	Rising edge at "Execute"

Tab. 5-128: Minimum / maximum values and default values of the IL_PNIOIoLinkIdentDetails inputs

Functional description

The function block **IL_PNIOIoLinkIdentDetails** is used to read identification parameters and device specific information of an IO-Link device connected to a desired IO-Link port.

The device is addressed via the PROFINET device name in "IdentStr". It must be part of the configuration of the controller. The data set on the PROFINET device is addressed via the Slot-SubSlot combination as well as via the PNIndex.

The function block reads the direct parameters of page 1 of an IOL device. Page 1 range from 16#00 to 16#0F. It comprises of communication protocol, identification parameters, and application control parameters. The function block reads identification parameters of page 1 and its subIndex range from 16#04 to 16#0D,

and it also reads the Indexes 16#10(VendorName), 16#11(VendorText), 16#12(ProductName), 16#13(ProductID), 16#14(ProductText), 16#15(SerialNumber), 16#16(HardwareRevision), 16#17(FirmwareRevision) and 16#18(ApplicationSpecificTag).

For non availability of optional; parameters default value empty string " " is displayed.

“Availability” variable confirms the Valid data available in the output by bit by bit as follows:

Availability Bit0 = TRUE => Index 16#00 (Direct Parameter Page 1) is available.

Availability Bit1 = TRUE => Index 16#10 (Vendor Name) is available.

Availability Bit2 = TRUE => Index 16#11 (Vendor Text) is available.

Availability Bit3 = TRUE => Index 16#12 (Product Name) is available.

Availability Bit4 = TRUE => Index 16#13 (Product ID) is available.

Availability Bit5 = TRUE => Index 16#14 (Product Text) is available.

Availability Bit6 = TRUE => Index 16#15 (Serial Number) is available.

Availability Bit7 = TRUE => Index 16#16 (Hardware Revision) is available.

Availability Bit8 = TRUE => Index 16#17 (Firmware Revision) is available.

Availability Bit9 = TRUE => Index 16#18 (Application Specific Tag) is available.

Example Program

This example project is configured to read the identification parameters of device BNI IOL-801-000-Z036 connected at port 6 of profinet gateway. The rising edge at "bExecute" input executes the read operation from the device and identification data is displayed at output attributes.

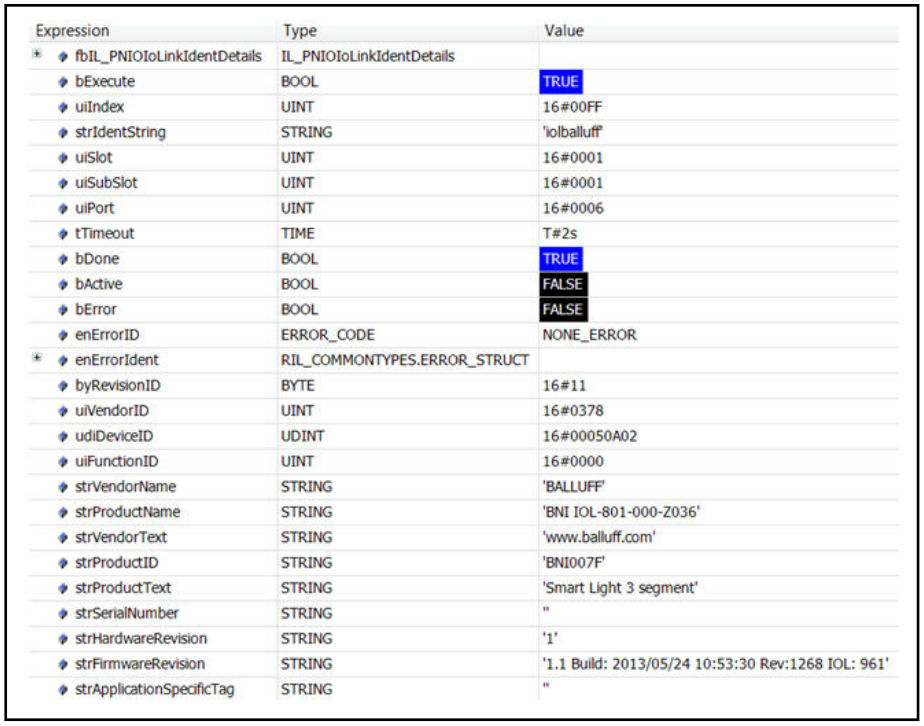
Declaration

```
PROGRAM ProgIL_PNIOIoLinkIdentDetails
VAR
    fbIL_PNIOIoLinkIdentDetails: IL_PNIOIoLinkIdentDetails;
```

```
    bExecute:         BOOL; //Function block is running when TRUE
    uiIndex:         UINT := 16#FF; //Index to read IOL data
    strIdentString:         STRING := 'iolballuff'; //Station name of PN BK
    uiSlot:            UINT := 16#01; //Addressing of the module slot.
                                // The slot addresses a logical or a physical module
    uiSubSlot:         UINT := 16#01; //Addressing of the submodule.
    uiPort:            UINT := 16#06; //IO-Link Port Number
    tTimeout:         TIME := T#2S; //Timeout monitoring of the function block
    bDone:             BOOL; //Function Block is completed its task
    bActive:          BOOL; //Function Block is working on its actual task
    bError:            BOOL; //Indicates an error
    enErrorID:         RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent:         RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
    wAvailable:        WORD;
    byRevisionID:         BYTE; // ID of the used protocol version for implementation. (XXXX YYYY of
                          //byte) the 'X' gives RevisionIDMajor and 'Y' gives RevisionIdMinor.
    uiVendorID:        UINT; // Unique vendor identification.
    udiDeviceID:        UDINT;    //Unique Device identification allocated BY a vendor.
    uiFunctionID:         UINT;    //Reserved (Engineering shall set both octets to "16#00")
    strVendorName:     STRING; // Name of the Vendor.
    strProductName:     STRING;// Name of the Product.
    strVendorText:         STRING;// Vendor text.
    strProductID:         STRING;// Unique Product identification allocated by a vendor.
    strProductText:         STRING; // Product Text.
    strSerialNumber:    STRING; // Vendor specific serial number.
    strHardwareRevision:     STRING; // Hardware revision of the Device.
    strFirmwareRevision:     STRING; // Firmware revision of the Device.
    strApplicationSpecificTag : STRING ; //It can serve as a "tag function" (role of the Device)
                                    // or a "tag location" (location of the Device).
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkIdentDetails(
    Execute:= bExecute ,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    PNIndex:= uiIndex,
    Slot:= uiSlot,
    Subslot:= uiSubSlot,
    Port:= uiPort,
    Timeout:= tTimeout,
    Available=> wAvailable,
    RevisionID=> byRevisionID,
    VendorID=> uiVendorID,
    DeviceID=> udiDeviceID,
    FunctionID=> uiFunctionID,
    VendorName=> strVendorName,
    ProductName=> strProductName,
    VendorText=> strVendorText,
    ProductID=> strProductID,
    ProductText=> strProductText,
    SerialNumber=> strSerialNumber,
    HardwareRevision=> strHardwareRevision,
    FirmwareRevision=> strFirmwareRevision,
    ApplicationSpecificTag=> strApplicationSpecificTag);
```



Expression	Type	Value
fbIL_PNIOIoLinkIdentDetails	IL_PNIOIoLinkIdentDetails	
bExecute	BOOL	TRUE
uiIndex	UINT	16#00FF
strIdentString	STRING	'iolballuff'
uiSlot	UINT	16#0001
uiSubSlot	UINT	16#0001
uiPort	UINT	16#0006
tTimeout	TIME	T#2s
bDone	BOOL	TRUE
bActive	BOOL	FALSE
bError	BOOL	FALSE
enErrorID	ERROR_CODE	NONE_ERROR
enErrorIdent	RIL_COMMONTYPES.ERROR_STRUCT	
byRevisionID	BYTE	16#11
uiVendorID	UINT	16#0378
udiDeviceID	UDINT	16#00050A02
uiFunctionID	UINT	16#0000
strVendorName	STRING	'BALLUFF'
strProductName	STRING	'BNI IOL-801-000-Z036'
strVendorText	STRING	'www.balluff.com'
strProductID	STRING	'BNI007F'
strProductText	STRING	'Smart Light 3 segment'
strSerialNumber	STRING	''
strHardwareRevision	STRING	'1'
strFirmwareRevision	STRING	'1.1 Build: 2013/05/24 10:53:30 Rev:1268 IOL: 961'
strApplicationSpecificTag	STRING	''

Fig. 5-46: Online execution of FB

All the IO-Link devices may not provide the complete identification data. Available data in connected device is processed and displayed at output attributes.

Troubleshooting

The function block uses error table **PROFINETIO_TABLE**(ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	ValueAdr is not set
INPUT_RANGE_ERROR	16#0001	16#0001	SizeOfValue is not set. SizeOfValue > 232 for Write request
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_RANGE_ERROR	16#0001	16#0003	IOLService is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header is not matching
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	internal use	State machine error

ErrorID	Additional1	Additional2	Remarks
Special Error Codes, IO Link Master (IOL-M errors)			
COMMUNICATION_ERROR	16#3000	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request
COMMUNICATION_ERROR	16#3000	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#3000	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#3000	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#3000	16#8001	Wrong index. IOL_Index 32767 ... 65535
COMMUNICATION_ERROR	16#3000	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#3000	16#8003	Wrong port function. Port function not supported
Special Error Codes, IO Link Device (application related errors)			
COMMUNICATION_ERROR	16#3001	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#3001	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#3001	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#3001	16#8023	Access denied
COMMUNICATION_ERROR	16#3001	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#3001	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#3001	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#3001	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#3001	16#8034	Parameter length under run
COMMUNICATION_ERROR	16#3001	16#8035	Function not available
COMMUNICATION_ERROR	16#3001	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#3001	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#3001	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#3001	16#8082	Application not ready
COMMUNICATION_ERROR	16#3001	16#8001..16#81FF	Vendor specific

Tab. 5-129: Error codes PNIO_IO-Link

5.7.15 IL_PNIOIoLinkRemoteDeviceState

Brief description The function block **IL_PNIOIoLinkRemoteDeviceState** reads out the current diagnosis status of the IO-Link device connected to a desired IO-Link port on profinet.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-130: Reference table

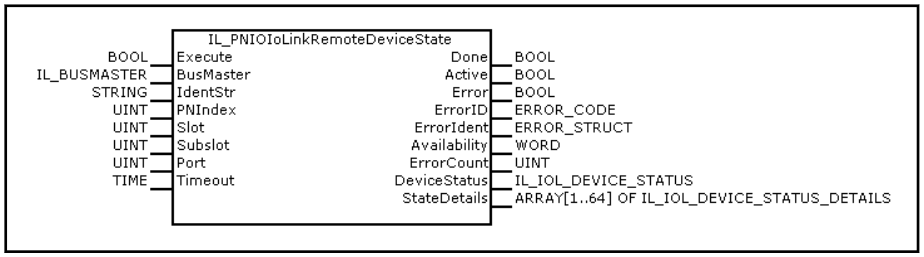


*Fig. 5-47: Interface Description of the **IL_PNIOIoLinkRemoteDeviceState** Function Block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the PROFINET device.
	PNIndex	UINT	255 – Regular index Call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example, an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	Port	UINT	1..63: IO-Link Port Number
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnosis in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable PROFINETIO_TABLE)

	Name	Type	Comment
	DeviceStatus	IL_IOL_DEVICE_STATUS	Information about the Device condition (diagnosis). Enumeration for device Status IL_IOL_DEVICE_STATUS, page 137
	StateDetails	ARRAY[1..64] OF IL_IOL_DEVICE_STATUS_DETAILS	Information about currently pending Events in the device. Structure for DeviceStatusDetails IL_IOL_DEVICE_STATUS_DETAILS, page 137

Tab. 5-131: FB IL_PNIOIoLinkRemoteDeviceState I/O interface

Min./Max. And Default Values: The following table describes the minimum, maximum and default values of the function block inputs.

Name	Type	Min. value	Max. Value	Default Value	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
IdentStr	STRING	---		''	Rising edge at "Execute"
PNIndex	UINT	0	16#ffff	255	Rising edge at "Execute"
Slot	UINT	0	16#7fff	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8fff	1	Rising edge at "Execute"
Port	UINT	0	255	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	----	T#2s	Rising edge at "Execute"

Tab. 5-132: Minimum / maximum values and default values of the IL_PNIOIoLinkRemoteDeviceState inputs

Functional description

The function block **IL_PNIOIoLinkRemoteDeviceState** reads out the current diagnosis status of the IO-Link device connected to a desired IO-Link port.

The device is addressed via the PROFINET device name in "IdentStr". It must be part of the configuration of the controller. The data set on the PROFINET device is addressed via the slot-subslot combination as well as via the "PNIndex". The function block uses Index 0x0020, 0x0024 and 0x0025 to access the "ErrorCount", "DeviceStatus" and "DeviceStatusDetails" respectively.

The parameter "ErrorCount" provides information on errors occurred in the device application since power-on or reset. The parameter detailed device Status shall provide information about currently pending events in the device. Device status parameters are optional in some of devices. For non available parameters default values assigned as follows

- ErrorCount: 0xFFFF (Default value)
- DeviceStatus: STATUS_INVALID (Default value)
- StateDetails: Empty (Default value).

"Availability" variable confirms the Valid data available in the output by bit by bit as follows:

- Availability Bit0 = TRUE => Index 0x20 (ErrorCount) is available
- Availability Bit1 = TRUE => Index 0x24 (Device Status) is available
- Availability Bit2 = TRUE => Index 0x25 (State Details) is available

Example Program

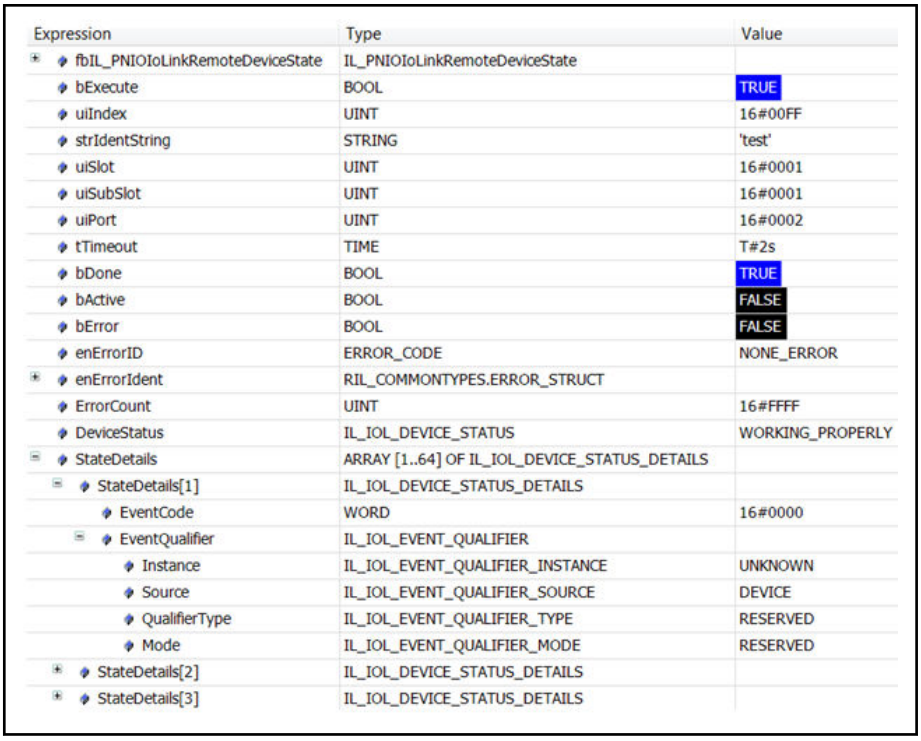
This example project is configured to read the identification parameters of device BFS 33M connected at port 2 of profinet gateway. The rising edge at "bExecute" input executes the read operation of Device Details from the device and "ErrorCount" and "DeviceState" is displayed at output attributes.

Declaration

```
PROGRAM ProgIL_PNIOIoLinkRemoteDeviceState
VAR
    fbIL_PNIOIoLinkRemoteDeviceState: IL_PNIOIoLinkRemoteDeviceState;
    bExecute:         BOOL; //Function block is running when TRUE
    uiIndex:         UINT := 16#FF; //Index to read IOL data
    strIdentString: STRING := 'test'; //Station name of PN BK
    uiSlot:     UINT := 16#01; //Addressing of the module slot.
                              // The slot addresses a logical or a physical module
    uiSubSlot:     UINT := 16#01; //Addressing of the submodule
    uiPort:         UINT := 16#02; //IO-Link Port Number
    tTimeout:     TIME := T#2S; //Timeout monitoring of the function block
    bDone:         BOOL; //Function Block is completed its task
    bActive:     BOOL; //Function Block is working on its actual task
    bError:         BOOL; //Indicates an error
    enErrorID:     RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent:     RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
    ErrorCount:     UINT := 0 ; //Information on errors occurred in the Device application
                              //since power-on or reset
    DeviceStatus:     IL_IOL_DEVICE_STATUS; //Information about the Device condition (diagnosis)
    StateDetails:     ARRAY[1..64] OF IL_IOL_DEVICE_STATUS_DETAILS ;
                              //Information about currently pending Events in the Device
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkRemoteDeviceState(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    PNIndex:= uiIndex,
    Slot:= uiSlot,
    Subslot:= uiSubSlot,
    Port:= uiPort,
    Timeout:= tTimeout,
    ErrorCount=> ErrorCount,
    DeviceStatus=> DeviceStatus,
    StateDetails=> StateDetails);
```



Expression	Type	Value
fbIL_PNIOIoLinkRemoteDeviceState	IL_PNIOIoLinkRemoteDeviceState	
bExecute	BOOL	TRUE
uiIndex	UINT	16#00FF
strIdentString	STRING	'test'
uiSlot	UINT	16#0001
uiSubSlot	UINT	16#0001
uiPort	UINT	16#0002
tTimeout	TIME	T#2s
bDone	BOOL	TRUE
bActive	BOOL	FALSE
bError	BOOL	FALSE
enErrorID	ERROR_CODE	NONE_ERROR
enErrorIdent	RIL_COMMONTYPES.ERROR_STRUCT	
ErrorCount	UINT	16#FFFF
DeviceStatus	IL_IOL_DEVICE_STATUS	WORKING_PROPERLY
StateDetails	ARRAY [1..64] OF IL_IOL_DEVICE_STATUS_DETAILS	
StateDetails[1]	IL_IOL_DEVICE_STATUS_DETAILS	
EventCode	WORD	16#0000
EventQualifier	IL_IOL_EVENT_QUALIFIER	
Instance	IL_IOL_EVENT_QUALIFIER_INSTANCE	UNKNOWN
Source	IL_IOL_EVENT_QUALIFIER_SOURCE	DEVICE
QualifierType	IL_IOL_EVENT_QUALIFIER_TYPE	RESERVED
Mode	IL_IOL_EVENT_QUALIFIER_MODE	RESERVED
StateDetails[2]	IL_IOL_DEVICE_STATUS_DETAILS	
StateDetails[3]	IL_IOL_DEVICE_STATUS_DETAILS	

Fig. 5-48: Online execution of FB

Troubleshooting

The function block uses error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	ValueAdr is not set
INPUT_RANGE_ERROR	16#0001	16#0001	SizeOfValue is not set. SizeOfValue > 232 for Write request
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_RANGE_ERROR	16#0001	16#0003	IOLService is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header is not matching
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	internal use	State machine error
Special Error Codes, IO Link Master (IOL-M errors)			
COMMUNICATION_ERROR	16#3000	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request

ErrorID	Additional1	Additional2	Remarks
COMMUNICATION_ERROR	16#3000	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#3000	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#3000	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#3000	16#8001	Wrong index. IOL_Index 32767 ... 65535
COMMUNICATION_ERROR	16#3000	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#3000	16#8003	Wrong port function. Port function not supported
Special Error Codes, IO Link Device (application related errors)			
COMMUNICATION_ERROR	16#3001	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#3001	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#3001	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#3001	16#8023	Access denied
COMMUNICATION_ERROR	16#3001	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#3001	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#3001	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#3001	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#3001	16#8034	Parameter length under run
COMMUNICATION_ERROR	16#3001	16#8035	Function not available
COMMUNICATION_ERROR	16#3001	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#3001	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#3001	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#3001	16#8082	Application not ready
COMMUNICATION_ERROR	16#3001	16#8001..16#81FF	Vendor specific

Tab. 5-133: Error codes PNIO_IO-Link

5.7.16 IL_PNIOIoLinkSetDeviceAccessLock

Brief description

The function block **IL_PNIOIoLinkSetDeviceAccessLock** allows controlling the Device behavior by overwriting the parameter Device Access Locks of IO-Link device connected on Profinet coupler. This parameter contains the following lock categories:

- Parameter write access (optional)
- Data Storage (mandatory if device supports data storage)
- Local parameterization (option)
- Local user interface operation (option)

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-134: Reference table

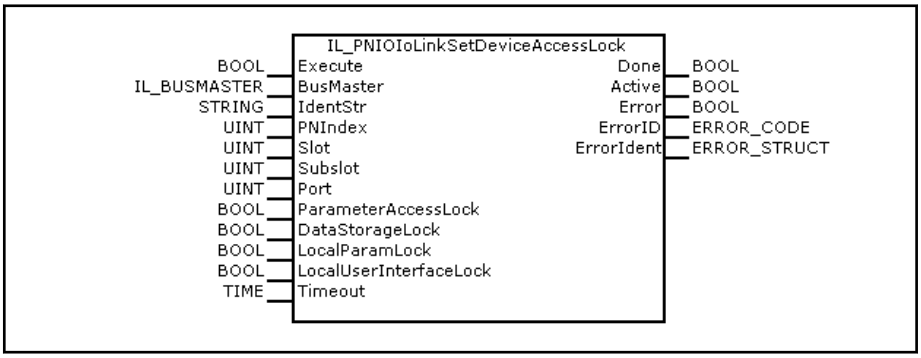


*Fig. 5-49: Interface Description of the **IL_PNIOIoLinkSetDeviceAccessLock** Function Block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the PROFINET device.
	PNIndex	UINT	255 – Regular index Call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example, an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	Port	UINT	1..63: IO-Link Port Number
	ParameterAccess-Lock	BOOL	Parameter write access lock status FALSE: Unlocked, TRUE: Locked
	DataStorageLock	BOOL	Data storage lock status FALSE: Unlocked, TRUE: Locked
	LocalParamLock	BOOL	Local parameterisation lock status FALSE: Unlocked, TRUE: Locked
	LocalUserInterface-Lock	BOOL	Local user interface lock status FALSE: Unlocked, TRUE: Locked
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid

	Name	Type	Comment
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnosis in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable PROFINETIO_TABLE)

Tab. 5-135: FB IL_PNIOIoLinkSetDeviceAccessLock I/O interface

Min./Max. And Default Values: The following table describes the minimum, maximum and default values of the function block inputs.

Name	Type	Min. value	Max. Value	Default Value	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING	---	---	---	Rising edge at "Execute"
PNIndex	UINT	0	16#ffff	0	Rising edge at "Execute"
Slot	UINT	0	16#7fff	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8fff	1	Rising edge at "Execute"
Port	UINT	0	255	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	----	T#2s	Rising edge at "Execute"
ParameterAc-cessLock	BOOL	FALSE	TRUE	FALSE	Rising edge at "Execute"
DataStorage-Lock	BOOL	FALSE	TRUE	FALSE	Rising edge at "Execute"
LocalParamLock	BOOL	FALSE	TRUE	FALSE	Rising edge at "Execute"
LocalUserInter-faceLock	BOOL	FALSE	TRUE	FALSE	Rising edge at "Execute"

Tab. 5-136: Minimum / maximum values and default values of the IL_PNIOIo-LinkSetDeviceAccessLock inputs

Functional description

The function block **IL_PNIOIoLinkSetDeviceAccessLock** allows controlling the Device behavior by overwriting the parameter Device Access Locks connected on desired IO-Link port. This function block writes IO-Link index 16#000C via Sub index 0. This parameter contains the following lock categories and support of these categories is device specific:

- Parameter write access (optional)
- Data Storage (mandatory if device supports data storage)
- Local parameterization (option)
- Local user interface operation (option)

If any category of this parameter is not supported by a connected device, Function block will return an error. The following table specifies the device locking possibilities.

The device is addressed via the PROFINET device name in "IdentStr", port number in "Port". It must be part of the configuration of the controller. The da-

ta set on the PROFINET device is addressed via the Slot-SubSlot combination as well as via the PNIndex.

The rising edge at Execute input starts the operation and after successful completion the output Done is set to TRUE. If any error then the Error bit is set and the detailed error information is available at ErrorID and ErrorIdent.

Device locking possibilities

Bit	Category	Definition
0	Parameter write access (optional)	0: unlocked (default) 1: locked
1	Data Storage (mandatory if device supports data storage)	0: unlocked (default) 1: locked (see Note)
2	Local parameterization (option)	0: unlocked (default) 1: locked
3	Local user interface operation (option)	0: unlocked (default) 1: locked
4-15	Reserved	
Note: The Master reads the parameter State_Properties/ State of Data Storage prior to any action		

Example Program

In this example project, DataStorageLock of Device Access Locks are over written to the IO-Link device connected on port 5.

The profinet bus coupler is available in slot 1 with "iolballuff" as station name.

The rising edge at "bExecute" input executes the operation and data of "ParameterAccessLock", "DataStorageLock", "LocalParamLock" and "LocalUserInterfaceLock" are written to the device.

Declaration

```
PROGRAM ProgIL_PNIOIoLinkSetDeviceAccessLock
VAR
    fb_PNIOIoLinkSetDeviceAccessLock: IL_PNIOIoLinkSetDeviceAccessLock; // FB Instance
    bExecute: BOOL; //Function block is running when TRUE
    uiIndex: UINT := 16#FF; //Index to read IOL data
    strIdentString: STRING := 'iolballuff'; //Station name of PN BK
    uiSlot: UINT := 16#01; //Addressing of the module slot.
                           //The slot addresses a logical or a physical module
    uiSubSlot: UINT := 16#01; //Addressing of the submodule.
    uiPort: UINT := 16#06; //IO-Link Port Number
    tTimeout: TIME := T#2S; //Timeout monitoring of the function block
    bDone: BOOL; //Function Block is completed its task
    bActive: BOOL; //Function Block is working on its actual task
    bError: BOOL; //Indicates an error
    enErrorID: RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
    bParameterAccessLock: BOOL; //Parameter write access lock
    bDataStorageLock: BOOL; //Data storage lock
    bLocalParamLock: BOOL; //Local parameterisation lock
    bLocalUserInterfaceLock: BOOL; //Local user interface lock
END_VAR
```

Implementation

```
fb_PNIOIoLinkSetDeviceAccessLock(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
```

```
BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
IdentStr:= strIdentString,
PNIndex:= uiIndex,
Slot:= uiSlot,
Subslot:= uiSubSlot,
Port:= uiPort,
ParameterAccessLock:= bParameterAccessLock,
DataStorageLock:= bDataStorageLock,
LocalParamLock:= bLocalParamLock,
LocalUserInterfaceLock:= bLocalUserInterfaceLock,
Timeout:= tTimeout);
```

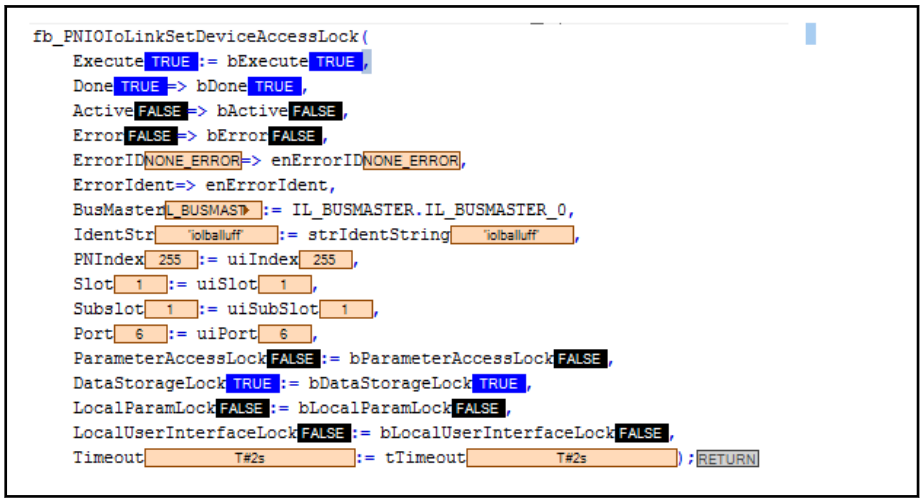


Fig. 5-50: Online execution of FB

Troubleshooting

The function block uses error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	ValueAdr is not set
INPUT_RANGE_ERROR	16#0001	16#0001	SizeOfValue is not set. SizeOfValue > 232 for Write request
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_RANGE_ERROR	16#0001	16#0003	IOLService is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header is not matching
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	internal use	State machine error
Special Error Codes, IO Link Master (IOL-M errors)			

ErrorID	Additional1	Additional2	Remarks
COMMUNICATION_ERROR	16#3000	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request
COMMUNICATION_ERROR	16#3000	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#3000	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#3000	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#3000	16#8001	Wrong index. IOL_Index 32767 ... 65535
COMMUNICATION_ERROR	16#3000	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#3000	16#8003	Wrong port function. Port function not supported
Special Error Codes, IO Link Device (application related errors)			
COMMUNICATION_ERROR	16#3001	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#3001	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#3001	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#3001	16#8023	Access denied
COMMUNICATION_ERROR	16#3001	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#3001	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#3001	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#3001	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#3001	16#8034	Parameter length under run
COMMUNICATION_ERROR	16#3001	16#8035	Function not available
COMMUNICATION_ERROR	16#3001	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#3001	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#3001	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#3001	16#8082	Application not ready
COMMUNICATION_ERROR	16#3001	16#8001..16#81FF	Vendor specific

Tab. 5-137: Error codes PNIO_IO-Link

5.7.17 IL_PNIOIoLinkSystemCommand

Brief description The function block **IL_PNIOIoLinkSystemCommand** used to write the system command to an IO-Link device connected to a desired IO-Link port on profinet.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-138: Reference table

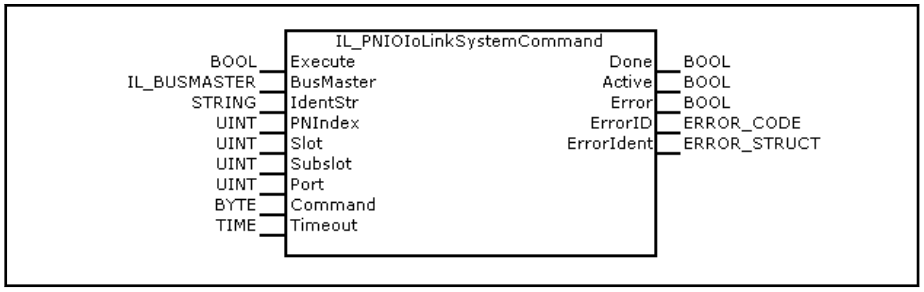


*Fig. 5-51: Interface Description of the **IL_PNIOIoLinkSystemCommand** Function Block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string of the PROFINET device.
	PNIndex	UINT	255 – Regular index Call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example, an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	Port	UINT	1..63: IO-Link Port Number
	Command	BYTE	System command.
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnosis in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable PROFINETIO_TABLE)

Tab. 5-139: FB IL_PNIOIoLinkSystemCommand I/O interface

Min./Max. And Default Values: The following table describes the minimum, maximum and default values of the function block inputs.

Name	Type	Min. value	Max. Value	Default Value	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"

Name	Type	Min. value	Max. Value	Default Value	Application
IdentStr	STRING	---		‘ ’	Rising edge at "Execute"
PNIndex	UINT	0	16#ffff	255	Rising edge at "Execute"
Slot	UINT	0	16#7fff	0	Rising edge at "Execute"
SubSlot	UINT	1	16#8fff	1	Rising edge at "Execute"
Port	UINT	0	255	0	Rising edge at "Execute"
Command	BYTE	0	255	0	Rising edge at "Execute"
Timeout	TIME	T#0ms	----	T#2s	Rising edge at "Execute"

Tab. 5-140: Minimum / maximum values and default values of the IL_PNIOIo-LinkSystemCommand inputs

Functional description

The function block **IL_PNIOIoLinkSystemCommand** shall write the system command to an IO-Link device connected to a desired IO-Link port. The device is addressed via the PROFINET device name in "IdentStr". It must be part of the configuration of the controller. The data set on the PROFINET device is addressed via the slot-subslot combination as well as via the PNIndex. Devices with ISDU support shall use the ISDU Index 0x0002 to receive the System Command.

A SystemCommand shall be executed within less than 5 s to fulfill the ISDU timing requirements.There are vendor specific SystemCommands also there.

The coding of system commands is specified in the table below.

Command (hex)	Command (dec)	Command name	M/O	Definition
0x00	0	Reserved		
0x01	1	ParamUploadStart	O	Start parameter upload
0x02	2	ParamUploadEnd	O	Stop parameter upload
0x03	3	ParamDownloadStart	O	Start parameter download
0x04	4	ParamDownloadEnd	O	Stop parameter download
0x05	5	ParamDownloadStore	O	Finalize parameterization and start Data Storage
0x06	6	ParamBreak	O	Cancel all Param commands
0x07 to 0x3F	7 to 63	Reserved		
0x40 to 0x7F	64 to 127	Reserved for profiles		
0x80	128	Device reset	O	
0x81	129	Application reset	O	
0x82	130	Restore factory settings	O	
0x83 to 0x9F	131 to 159	Reserved		
0xA0 to 0xFF	160 to 255	Vendor specific		
NOTE See 10.3				
Key M = mandatory; O = optional				

Fig. 5-52: Coding of System Commands

Device reset: This feature enables a Device to perform a "warm start". It is especially useful whenever a Device has to be reset to an initial state such as power-on.

Application reset: This feature enables a Device to reset the technology specific application. It is especially useful whenever a technology specific application has to be set to a predefined operational state without communication interruption and a shut-down cycle.

Factory settings: This feature enables a Device to restore parameters to the original delivery status.

Example Program

In this example project, FB IL_PNIOIoLinkSystemCommand is configured and executed to Restore factory settings for the IO-Link device connected on port 5.

The profinet bus coupler is available in slot 1 with 'iolballuff' as station name.

The rising edge at "bExecute" input executes the operation, this writes the system command Restore factory settings - 16#82 to the device defined in input "ByCommand".

Declaration

```
PROGRAM ProgIL_PNIOIoLinkSystemCommand
VAR
    fbIL_PNIOIoLinkSystemCommand: IL_PNIOIoLinkSystemCommand;
    bExecute:         BOOL; //Function block is running when TRUE
    uiIndex:         UINT := 16#FF; //Index to read IOL data
    strIdentString: STRING := 'iolballuff'; //Station name of PN BK
    uiSlot:         UINT := 16#01; //Addressing of the module slot.
                                   //The slot addresses a logical or a physical module
    uiSubSlot:         UINT := 16#01; //Addressing of the submodule.
    uiPort:         UINT := 16#06; //IO-Link Port Number
    ByCommand:        BYTE :=0; //System command.
    tTimeout:         TIME := T#2S; //Timeout monitoring of the function block
    bDone:             BOOL; //Function Block is completed its task
    bActive:         BOOL; //Function Block is working on its actual task
    bError:         BOOL; //Indicates an error
    enErrorID:         RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent:     RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkSystemCommand(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    PNIndex:= uiIndex,
    Slot:= uiSlot,
    SubSlot:= uiSubSlot,
    Port:= uiPort,
    Command:= ByCommand,
    Timeout:= tTimeout);
```

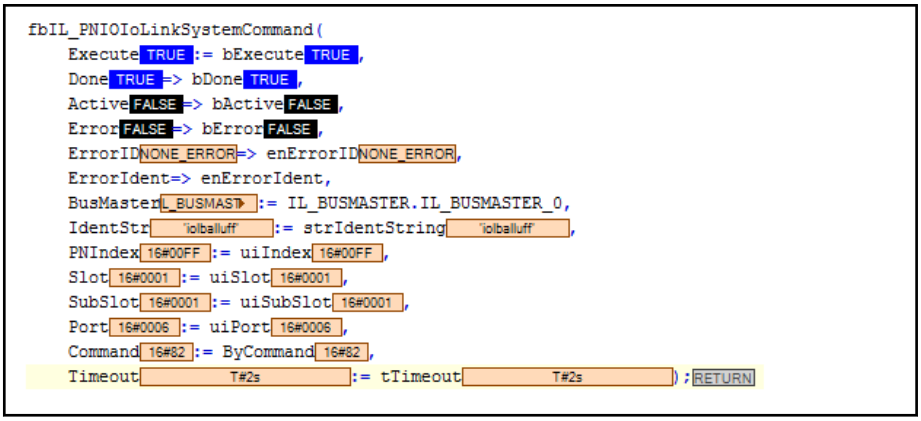


Fig. 5-53: Online execution of FB

Troubleshooting

The function block uses error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0000	ValueAdr is not set
INPUT_RANGE_ERROR	16#0001	16#0001	SizeOfValue is not set. SizeOfValue > 232 for Write request
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
INPUT_RANGE_ERROR	16#0001	16#0003	IOLService is invalid
COMMUNICATION_ERROR	16#0003	*PROFINET IO Controller Diagnosis* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
COMMUNICATION_ERROR	16#0005	16#0000	Response header is not matching
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	timeout
STATE_MACHINE_ERROR	16#0105	internal use	State machine error
Special Error Codes, IO Link Master (IOL-M errors)			
COMMUNICATION_ERROR	16#3000	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request
COMMUNICATION_ERROR	16#3000	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#3000	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#3000	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#3000	16#8001	Wrong index. IOL_Index 32767 ... 65535
COMMUNICATION_ERROR	16#3000	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#3000	16#8003	Wrong port function. Port function not supported

ErrorID	Additional1	Additional2	Remarks
Special Error Codes, IO Link Device (application related errors)			
COMMUNICATION_ERROR	16#3001	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#3001	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#3001	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#3001	16#8023	Access denied
COMMUNICATION_ERROR	16#3001	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#3001	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#3001	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#3001	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#3001	16#8034	Parameter length under run
COMMUNICATION_ERROR	16#3001	16#8035	Function not available
COMMUNICATION_ERROR	16#3001	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#3001	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#3001	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#3001	16#8082	Application not ready
COMMUNICATION_ERROR	16#3001	16#8001..16#81FF	Vendor specific

Tab. 5-141: Error codes PNIO_IO-Link

5.7.18 IL_PNIOIoLinkBackupDevice

Brief description The function block **IL_PNIOIoLinkBackupDevice** is used to back up the configuration and parameters of an IOL device (operated as a Profinet IO-Link gateway) to a file.

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/IO-Link
MTX	RIL_ProfinetIO/POUs/IO-Link

Tab. 5-142: Reference table

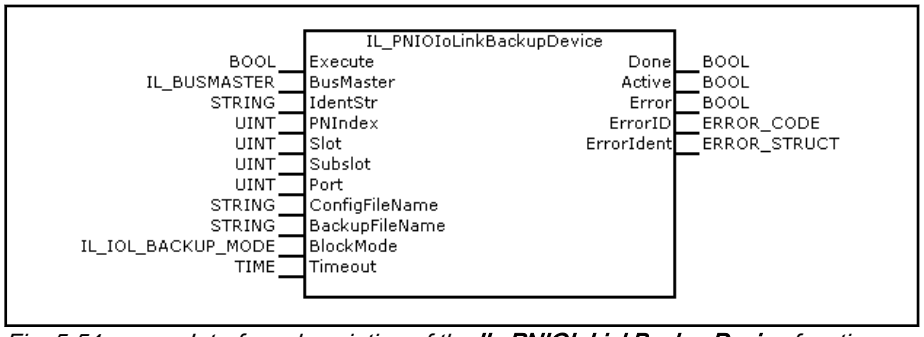


*Fig. 5-54: Interface description of the **IL_PNIOIoLinkBackupDevice** function block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 184
	IdentStr	STRING	Identification string of the Profinet device.
	PNIndex	UINT	255 – Regular index call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing the module slot. The slot addresses a logical or a physical module (for example an I/O module)
	SubSlot	UINT	Addressing the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot).
	Port	UINT	1..63: IO-Link, port number
	ConfigFileName	STRING	ConfigFile with index and subindex to read the parameters from the device.
	BackupFileName	STRING	Backup file created/overwritten by the function block.
	BlockMode	IL_IOL_BACK-UP_MODE	Enumeration for the “IOL Block” mode. Refer to IL_IOL_BACKUP_MODE, page 139
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnostics in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE)

Tab. 5-143: I/O interface of the IL_PNIOIoLinkBackupDevice function block

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	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING	---	---	“”	Rising edge at "Execute"
PNIndex	UINT	16#0	16#FFFF	16#FF	Rising edge at "Execute"
Slot	UINT	16#0	16#7FFF	16#0	Rising edge at "Execute"

Name	Type	Min. value	Max. value	Default value	Application
SubSlot	UINT	16#1	16#8FFF	16#1	Rising edge at "Execute"
Port	UINT	16#0	16#FF	16#0	Rising edge at "Execute"
ConfigFileName	STRING	---	---	''	Rising edge at "Execute"
BackupFileName	STRING	---	---	''	Rising edge at "Execute"
BlockMode	IL_IOL_BLOCK_MODE	NO_BLOCK	BLOCK	N0_BLOCK	Rising edge at "Execute"
Timeout	TIME	T#0ms		T#2s	Rising edge at "Execute"

Tab. 5-144: Min., max. and default values of the IL_PNIOIoLinkBackupDevice inputs

Functional description

The function block **IL_PNIOIoLinkBackupDevice** is used to back up the configuration and parameters of an IOL device (operated as a Profinet IO-Link gateway) to a file. The function block retrieves the parameters listed in the configuration file "ConfigFileName" from the device and saves the data to the file "BackupFileName". The configuration files and the backup files are stored in the flash memory of the controller.

The function block creates a new file "BackupFileName" upon each execution. If the file already exists, it is overwritten.

Index, subindex	16#0001,16#05
Index, subindex	16#0020,16#02
Index, subindex	16#0022,16#00
...	...
...	...
...	...
Index, subindex	16#0042,16#01
Basic structure of the ConfigFile	**Example of the ConfigFile**

The function block uploads the parameters in Block/No-Block modes as selected at the input "BlockMode".

1. **NO_BLOCK**: Uploading parameters without parameter blocking. Parameters are read individually.
2. **BLOCK**: Uploading parameters with parameter blocking. Before uploading data, the command "ParamUploadStart" is executed. After backing up data, the command "ParamUploadEnd" is executed. In this case, the user can inhibit write requests from other sources to ensure data integrity.

At the rising edge at the"Execute" input, the function block reads the parameters from the device and stores the data in the backup file according to the selected mode.

VendorName:	VendorName: XYZ
VendorID:	VendorID: XYZ
ProductName:	ProductName: XYZ
DeviceID:	DeviceID: XYZ
RevisionID:	RevisionID: XYZ
[PARAMETER_VALUES]	[PARAMETER_VALUES]
Index, SubIndex, DataSize, Data	16#0002,16#01,16#01,16#1B
Index, SubIndex, DataSize, Data1, Data2, Data3	16#0005,16#00,16#03,16#28,16#2F,16#43
...	...
...	...
Index, SubIndex, DataSize, Data	16#0020,16#03,16#01,16#08
Basic backup file	**Example of the backup file**

Example program

This example project is configured to back up the parameters specified in "ConfigFileName" and to store them on the file named "BackupFileName".

The identification parameters of the device BNI IOL-801-000-Z036 are connected at port 6 of the Profinet gateway. At a rising edge at the "bExecute" input, the function block executes the backup operation from the device. The result is displayed at the output attributes. The configuration file has to be stored on the user partition of the file system of the control. To transfer the file, use the Device Explorer in IndraWorks. Right-click on the icon of the control to open the following menu.

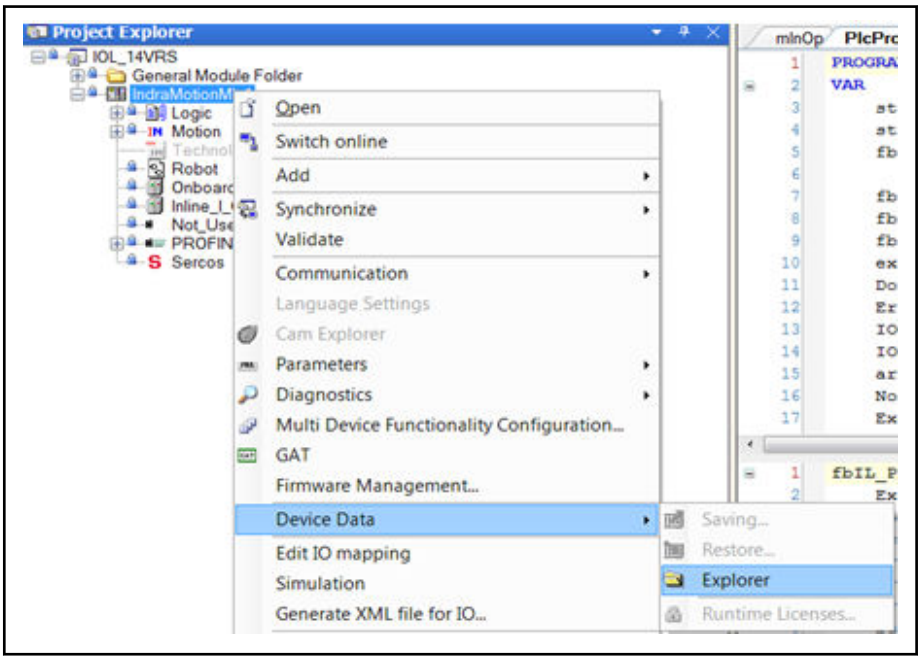


Fig. 5-55: Selecting the Device Explorer

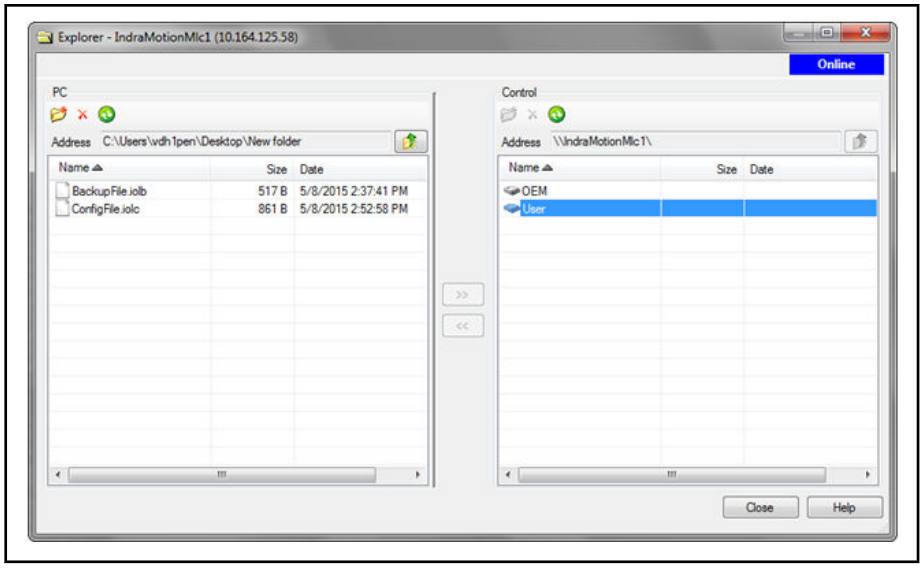


Fig. 5-56: Selecting the user partition

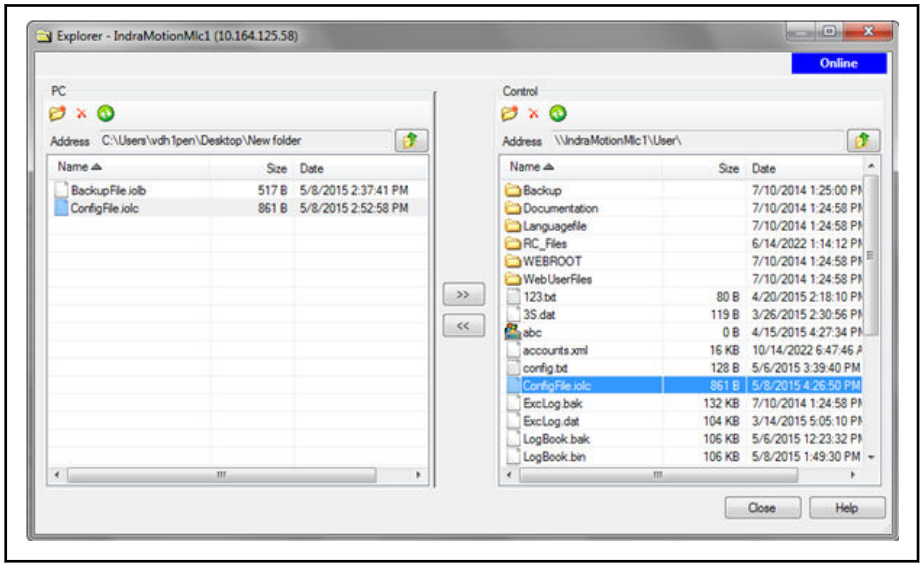


Fig. 5-57: Selecting configuration file

Declaration

```
PROGRAM ProgIL_PNIOIoLinkBackupDevice
VAR
    fbIL_PNIOIoLinkBackupDevice: IL_PNIOIoLinkBackupDevice; // FB Instance

    bExecute:           BOOL; //Function block is running when TRUE
    uiIndex:            UINT := 255; // PN Index
    strIdentString:     STRING := 'iolballuff'; //Station name of PN BK
    uiSlot:             UINT := 16#01; //Addressing of the module slot.
                               //The slot addresses a logical or a physical module
    uiSubSlot:          UINT := 16#01; //Addressing of the submodule.
    uiPort:             UINT := 16#06; //IO-Link Port Number
    strConfigFileName: STRING:= 'ConfigFile'; // Configuration file name
    strBackupFileName: STRING:= 'BackupFile'; // Backup file name
    enBlockMode:        IL_IOL_RESTORE_MODE := IL_IOL_BACKUP_MODE.BLOCK; // Backup Mode
    tTimeout:           TIME := T#3S; //Timeout monitoring of the function block
    bDone:              BOOL; //Function Block is completed its task
    bActive:            BOOL; //Function Block is working on its actual task
    bError:             BOOL; //Indicates an error
    enErrorID:          RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent:       RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkBackupDevice(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    PNIndex:= uiIndex,
    Slot:= uiSlot,
    SubSlot:= uiSubSlot,
    Port:= uiPort,
    ConfigFileName:= strConfigFileName,
    BackUpFileName:= strBackupFileName,
    BlockMode:= enBlockMode,
    Timeout:= tTimeout);
```

Troubleshooting The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
COMMUNICATION_ERROR	16#0003	*Profinet I/O controller diagnostics* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	
COMMUNICATION_ERROR	16#0004	Profinet I/O status • Special error codes, Profinet IO status , page 193	
SYSTEM_ERROR	16#0101	Internal use	Controller not found in configuration table
COMMUNICATION_ERROR	16#0104	16#0000	Timeout
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	SubIndex not available
COMMUNICATION_ERROR	16#3001	16#8023	Access denied, the device does not support the BlockMode
INPUT_INVALID_ERROR	16#3002	16#0000	"BlockMode" input is invalid
INPUT_INVALID_ERROR	16#3002	16#0001	"ConfigFileName" is invalid
INPUT_INVALID_ERROR	16#3002	16#0002	"BackupFileName" is invalid
ACCESS_ERROR	16#3003	16#0000	"Configuration file" could not be read
ACCESS_ERROR	16#3003	16#0001	"Configuration file" could not be opened
ACCESS_ERROR	16#3003	16#0002	The "Configuration file" format is invalid
ACCESS_ERROR	16#3003	16#0003	"Backup file" could not be opened
ACCESS_ERROR	16#3003	16#0004	Reading/writing the "backup file" is not possible

Tab. 5-145: Error codes of the IL_PNIOIoLinkBackupDevice

5.7.19 IL_PNIOIoLinkRestoreDevice

Brief description The function block **IL_PNIOIoLinkRestoreDevice** is used to restore the parameters of an IOL device from the backup file via the Profinet I/O link gateway.

Target system	Library/folder
MLC	RIL_ProfinetIO/ POUs/ IO-Link
MTX	RIL_ProfinetIO/ POUs/ IO-Link

Tab. 5-146: Reference table

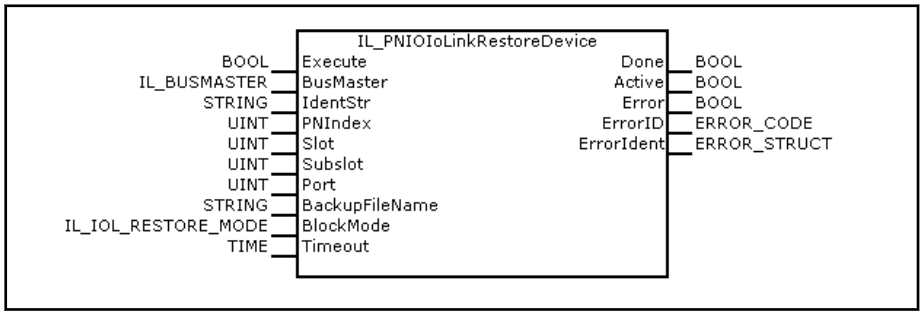


*Fig. 5-58: Interface description of the **IL_PNIOIoLinkRestoreDevice** function block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the controller in the order of the configuration tree. If only one Profinet I/O controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 184
	IdentStr	STRING	Identification string of the Profinet device
	PNIndex	UINT	255 – Regular index call 0...254 - Index redirection (cap)
	Slot	UINT	Addressing of the module slot. The slot addresses a logical or a physical module (for example an I/O module)
	SubSlot	UINT	Addressing of the submodule. A module can consist of several logical or physical submodules. At least the submodule with number 1 is present in each module (slot)
	Port	UINT	1..63: I/O link port number
	BackupFileName	STRING	Backup file generated/overwritten by the function block.
	BlockMode	IL_IOL_RESTORE_MODE	Enumeration for IOL Block parameter mode. Refer IL_IOL_RESTORE_MODE, page 139
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing not yet completed, output data invalid
	Error	BOOL	Processing completed with error

	Name	Type	Comment
	ErrorID	ERROR_CODE	Describes the diagnostics in the event of an error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE)

Tab. 5-147: Function block IL_PNIOIoLinkRestoreDevice I/O interface

Min./max. and default values: The following table describes the min., max. and default values of the function block inputs.

Name	Type	Min. value	Max. value	Default value	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING	---	---	---	Rising edge at "Execute"
PNIndex	UINT	16#0	16#FFFF	16#FF	Rising edge at "Execute"
Slot	UINT	16#0	16#7FFF	16#0	Rising edge at "Execute"
SubSlot	UINT	16#1	16#8FFF	16#1	Rising edge at "Execute"
Port	UINT	16#0	16#FF	16#0	Rising edge at "Execute"
BackupFile-Name	STRING	---	---	---	Rising edge at "Execute"
BlockMode	IL_IOL_RE-STORE_MODE	BLOCK_CHEC K	NO_BLOCK_N O_CHECK	BLOCK_CHE CK	Rising edge at "Execute"
Timeout	TIME	T#0ms	---	T#2s	Rising edge at "Execute"

Tab. 5-148: Min., max. and default values of the IL_PNIOIoLinkRestoreDevice inputs

Functional description

The function block **IL_PNIOIoLinkRestoreDevice** is used to restore the parameters of an IOL device from the backup file via the Profinet I/O link gateway. The function block reads the parameters from the IOL device backup file stored in the CF memory of the controller. The parameters can be restored in three modes as specified in the "IL_IOL_RESTORE_MODE".

1. **BLOCK_CHECK**: Downloading parameters with parameter blocking and with validity check. Before restoring data, the command "ParamDownloadStart" is executed. After restoring data, the command "ParamDownloadStore" is executed. In this case, writing parameters from other sources is inhibited and the validity check command is initiated after the last parameter writing.
2. **NO_BLOCK_CHECK**: Downloadingparameters without parameter blocking and with online validity check. After restoring data, the command "ParamDownloadStore" is executed initiating the validity check after the last parameter writing.
3. **NO_BLOCK_NO_CHECK**: Downloading of parameters without parameter blocking and validity check. Only single parameter writing is performed. The parameter validity check is executed internally by the device upon each parameter writing.

The variable at the input "BackupFileName" should only consist of the file name without extension.

At a rising edge at "Execute" the function block reads all parameters from the backup file and writes the parameter values to the device using the selected restoration mode.

Example program

This is the example project to restore the IOL device parameters from backup file stored in the CF card in the IndraControl device.

Define the restoration method in "enBlockMode". Specify the backup file name in "strFileName". Enter the file name without any extension, e.g. "Backupfile". The rising edge the at "bExecute" input executes the operation by restoring the parameters available in backup file to the device specified in "uiPort".

Declaration

```
PROGRAM ProgIL_PNIOIoLinkRestoreDevice
VAR
    fbIL_PNIOIoLinkRestoreDevice: IL_PNIOIoLinkRestoreDevice; // FB Instance

    bExecute:       BOOL; //Function block is running when TRUE
    uiIndex:        UINT := 255; // PN Index
    strIdentString: STRING := 'iolballuff'; //Station name of PN BK
    uiSlot:         UINT := 16#01; //Addressing of the module slot.
                                   //The slot addresses a logical or a physical module
    uiSubSlot:      UINT := 16#01; //Addressing of the submodule.
    uiPort:         UINT := 16#06; //IO-Link Port Number
    strFileName:    STRING:= 'BackupFile'; // Backup file name
    enBlockMode:    IL_IOL_RESTORE_MODE := IL_IOL_RESTORE_MODE.BLOCK_CHECK; // Backup Mode
    tTimeout:       TIME := T#3S; //Timeout monitoring of the function block
    bDone:          BOOL; //Function Block is completed its task
    bActive:        BOOL; //Function Block is working on its actual task
    bError:         BOOL; //Indicates an error
    enErrorID:      RIL_COMMONTYPES.ERROR_CODE; //Class of error
    enErrorIdent:   RIL_COMMONTYPES.ERROR_STRUCT; //Detailed information about error
END_VAR
```

Implementation

```
fbIL_PNIOIoLinkRestoreDevice(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    IdentStr:= strIdentString,
    PNIndex:= uiIndex,
    Slot:= uiSlot,
    SubSlot:= uiSubSlot,
    Port:= uiPort,
    BackUpFileName:= strFileName,
    BlockMode:= enBlockMode,
    Timeout:= tTimeout);
```

Troubleshooting

The function block uses error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Remarks
INPUT_RANGE_ERROR	16#0001	16#0002	IdentStr is invalid
COMMUNICATION_ERROR	16#0003	*Profinet I/O controller diagnostics* • General Error Codes , page 191 • Special Error Codes, Profinet I/O Controller, page 193	

ErrorID	Additional1	Additional2	Remarks
COMMUNICATION_ERROR	16#0004	PROFINET IO Status • Special error codes, Profinet IO status , page 193	
SYSTEM_ERROR	16#0101	internal use	Controller not found in Configuration table
COMMUNICATION_ERROR	16#0104	16#0000	Timeout
COMMUNICATION_ERROR	16#3001	16#8011	Index not available
COMMUNICATION_ERROR	16#3001	16#8012	Subindex not available
COMMUNICATION_ERROR	16#3001	16#8023	Access denied/ DataStorage locked/ Block mode is not supported by device
INPUT_INVALID_ERROR	16#3002	16#0000	BlockMode input is invalid.
INPUT_INVALID_ERROR	16#3002	16#0002	BackupFileName is invalid.
ACCESS_ERROR	16#3003	16#0002	The format of BackUp file is invalid
ACCESS_ERROR	16#3003	16#0003	The backup file can not be opened
ACCESS_ERROR	16#3003	16#0004	The backup file can not be Read/write

Tab. 5-149: Error codes IL_PNIOIoLinkBackupDevice

5.8 Data types

5.8.1 IL_BUSMASTER

Brief description

The IL_BUSMASTER enumeration type of the RIL_FieldbusTypes.library is used to select the respective Profinet I/O controller. In addition to the onboard real-time Ethernet interface, other controllers can also be available in an MLC system via function modules.

The Profinet I/O controllers are distinguished based on their ascending order in the configuration.

Indexes are assigned to the controllers in the following order:

- Onboard master
- Function module 1,
- Function module 2,
- Function module 3,
- Function module 4

☞ If, for example, the onboard master is not a PROFINET IO controller while a PROFINET IO controller is configured on the FM1, IL_BUSMASTER_0 is the value assigned to the controller on the FM1.

Element	Value	Description
IL_BUSMASTER_0	0	First PN controller
IL_BUSMASTER_1	1	Second PN controller
IL_BUSMASTER_2	2	Third PN controller
IL_BUSMASTER_3	3	Fourth PN controller
IL_BUSMASTER_4	4	Fifth PN controller

Tab. 5-150: IL_BUSMASTER enumeration type

5.8.2 IL_FBUS_COMMUNICATION_STATE

Brief description The enumeration type **IL_FBUS_COMMUNICATION_STATE** is used to display the status of the field bus communication.

Element	Value	Description
COMM_STATE_NOT_CONFIGURED	16#00000001	Communication is not configured
COMM_STATE_STOP	16#00000002	Communication is in "Stop" state
COMM_STATE_IDLE	16#00000003	Communication is in "Idle" state
COMM_STATE_OPERATE	16#00000004	Communication in cyclic operation

Tab. 5-151: IL_FBUS_COMMUNICATION_STATE enumeration type

5.8.3 IL_FBUS_SLAVE_STATE

The enumeration type **IL_FBUS_SLAVE_STATE** is used to show the status of the communication relations to the slaves. That means, that the status indicates whether the field bus master has an active connection to all configured slaves.

Element	Value	Description
SLAVE_STATE_OK	16#00000001	Communication to all active slaves available
SLAVE_STATE_FAILED	16#00000002	Communication error to at least one slave

Tab. 5-152: IL_FBUS_SLAVE_STATE enumeration slave

5.8.4 IL_PNIO_CONTROLLER_STATE

Brief description Data structure IL_PNIO_CONTROLLER_STATE contains diagnostic information about

- state of the communication stack,
- the state of the bus communication,
- error counters,
- an overview on configured, active slaves reporting a diagnostics

Structural element	Type	Description
CommunicationCOS	DWORD	Status of the communication stack, see below
CommunicationState	IL_FBUS_COMMUNICATION_STATE	Status of the communication See IL_FBUS_COMMUNICATION_STATE, page 185.
Version	UINT	Version of the data structure of the diagnostic information
Watchdog	UINT	Watchdog timeout between PLC and field bus connection in ms
ErrorCount	UDINT	Error counter. Counts the number communication errors that have occurred (both internal errors and communication errors)
SlaveState	IL_FBUS_SLAVE_STATE	Indicates whether the PROFINET IO controller has an active connection to all configured slaves. See IL_FBUS_SLAVE_STATE, page 185.
NumOfConfigSlaves	UDINT	Number of configured slaves

Structural element	Type	Description
NumOfActiveSlaves	UDINT	Number of active slaves, i.e., slaves to which the master has established a communication connection.
NumOfDiagSlaves	UDINT	Number of slaves with an active diagnostics

Tab. 5-153: Data structure IL_PNIO_CONTROLLER_STATE

Structural element CommunicationCOS

Bit no.	Name	Description
0	COMM_COS_READY	Protocol stack has been started and waits for configuration
1	COMM_COS_RUN	Protocol stack has been configured and waits for bus connection
2	COMM_COS_BUS_ON	Protocol stack may communicate on the bus
3	COMM_COS_CONFIG_LOCKED	Bus configuration is locked and cannot be changed
4	COMM_COS_CONFIG_NEW	New bus configuration is available but not initialized yet
5	COMM_COS_RESTART_REQUIRED	Communication restart required
6	COMM_COS_RESTART_REQUIRED_ENABLE	Communication restart possible
		FALSE assigned to remaining ones

The state of the communication stack is binary-encoded in the CommunicationCOS output.

The bits are contained in the RIL_FIELDBUS_TYPES library in the form of defines (PNIO_Defines).

5.8.5 IL_PNIO_PORT_DATA

Brief description The data structure **IL_PNIO_PORT_DATA** contains information on interconnected ports of neighboring Profinet I/O devices.

Structural element	Value	Description
DeviceIdx	UINT	Internal use
PeerPortIdx	UINT	Internal use
OwnPortIDStr	STRING(15)	Name of the own port
PeerPortIDStr	STRING(15)	Name of the removed port connected to the own port (Port name of the neighbor)
PeerChassisIDStr	STRING(255)	Name of the removed Profinet I/O device connected to the own Profinet I/O device (Device name of the neighbor)
PeerMACAddress	ARRAY[0..5] OF BYTE	MAC address of the removed Profinet I/O device

Tab. 5-154: IL_PNIO_PORT_DATA structure

5.8.6 IL_PNIO_REMOTE_DEVICE_INFO

Brief description The data structure contains a summary of the Profinet state and diagnostic information on a remote device.

Structural element	Type	Description
DeviceName	STRING(255)	StationName ("IdentStr")
DeviceID	UINT	Device ID
VendorID	UINT	Vendor ID
DeviceState	IL_PNIO_REMOTE_DEVICE_STATE	Device state chapter 5.8.7 "IL_PNIO_REMOTE_DEVICE_STATE" on page 187
DiagInfo	PNIO_DIAGINFO	Detailed device diagnostics chapter 5.8.11 "PNIO_DIAGINFO" on page 189

Tab. 5-155: IL_PNIO_REMOTE_DEVICE_INFO structure

5.8.7 IL_PNIO_REMOTE_DEVICE_STATE

Brief description The data structure contains a summary of the simplified device status of a remote device.

Structural element	Type	Description
RemoteDeviceNotReachable	BIT	0.0, device not reachable
RemoteDeviceNotReady	BIT	0.1, device not ready
RemoteDeviceConfigError	BIT	0.2, device configuration error
RemoteDeviceDiagnosis	BIT	0.3, device diagnostics
RemoteDeviceError	BIT	0.4, device errors
Reserved05	BIT	0.5
Reserved06	BIT	0.6
Reserved07	BIT	0.7
Reserved10	BIT	1.0
...	BIT	...
Reserved17	BIT	1.7

Tab. 5-156: IL_PNIO_REMOTE_DEVICE_STATE structure

5.8.8 IL_PNIO_TOPO_STATE

Brief description The data structure **IL_PNIO_TOPO_STATE** contains information on the topology of a Profinet I/O device.

The saved topology is compared to the current topology.

Structural element	Type	Description
OwnPortChanged	Bit	0.0 (16#0001) own port(s) changed or exchanged
NeighborDiff_1	Bit	0.1 (16#0002) one neighbor not as expected
NeighborDiff_2m	Bit	0.2 (16#0004) two or more neighbors not as expected
NeighborMiss_1	Bit	0.3 (16#0008) one neighbor not available
NeighborMiss_2m	Bit	0.4 (16#0010) two or more neighbors not available
ReadError	Bit	0.5 (16#0020) reading error, Profinet I/O device cannot be reached

Structural element	Type	Description
Reserved06	BIT	0.6 (16#0040)
...		
Reserved17	BIT	1.7 (16#8000)

Tab. 5-157: IL_PNIO_TOPO_STATE structure

5.8.9 IL_PNIO_TOPO_STATUS

Brief description The data structure **IL_PNIO_TOPO_STATUS** contains information on the status of a Profinet I/O device.

Structural element	Value	Description
Device	PNIO_LiveList	Contains information on a Profinet I/O device chapter 5.8.13 "PNIO_LIVELIST" on page 189
TopoState	IL_PNIO_TOPO_STATE	Contains information on the environment of a Profinet I/O device (difference between the saved and the current topology) chapter 5.8.8 "IL_PNIO_TOPO_STATE" on page 187
NoOfPorts	USINT	Number of ports
NoOfNeighborsStored	USINT	Number of saved neighbors (PNTopo_n.xml)
NoOfNeighborsActual	USINT	Number of currently read neighbors
NoOfNeighborsDiff	USINT	Number of neighbors with differences between the saved and the read topology

Tab. 5-158: IL_PNIO_TOPO_STATUS structure

5.8.10 PNIO_CHANNELERROR

Brief description Enumeration type PNIO_ CHANNELERROR generates the ChannelErrorType according to IEC 61158-6-10 V23 6.2.7.2.

There may be additional manufacturer-specific values.

Element	Value	Description
ChanErr_Unknown	0	Unknown error
ChanErr_ShortCircuit	1	Short circuit
ChanErr_Undervoltage	2	Voltage too low
ChanErr_Overvoltage	3	Voltage too high
ChanErr_Overtemperature	4	Overtemperature
ChanErr_Overload	5	Overload
ChanErr_LineBreak	6	Cable Break
ChanErr_UpperLimit	7	Value range exceeded
ChanErr_LowerLimit	8	Value range fallen below
ChanErr_Error	9	General error
ChanErr_SimulationActive	10	Simulation mode

Tab. 5-159: Enumeration type PNIO_CHANNELERROR

5.8.11 PNIO_DIAGINFO

Brief description The data structure contains a summary of the PROFINET state and diagnostic information. It consists of the type (DiagType), the location (slot, subslot, channel) and, if necessary, an error type of the event.

Structural element	Type	Description
DiagType	PNIO_DIAGTYPE	Event type
Slot	UINT	Slot where the event has occurred.
Subslot	UINT	Subslot where the event has occurred.
Channel	UINT	Channel in which the event occurred, 16#FFFF if no channel is available. Here, a channel is the individual I/O point which may deliver a diagnosis (e.g., cable break).
ChannelErrorType	PNIO_CHANNELERROR	Event or diagnosis
ExtErrorType	UINT	Extended, vendor-specific error type
AddDiagInfo	UDINT	Additional diagnostic information

Tab. 5-160: Structure of PNIO_DIAGINFO

5.8.12 PNIO_DIAGTYPE

Brief description The PNIO_DIAGTYPE enumeration type serves to label the events that can be diagnosed.

☞ Surplus modules may be present for PROFINET devices. These modules do not generate any diagnostic message.

Element	Value	Description
NO_DIAGNOSIS	0	No event/diagnosis
DIAGNOSIS_APPEARS	1	There is a diagnostic message. Evaluate PNIO_CHANNELERROR
MODULE_MISSING	2	The module designated by Slot/Subslot is configured but not present at the device
MODULE_WRONG	3	The module designated by Slot/Subslot does not correspond to the configured module
SUBMODULE_STATE	4	The module designated by Slot/Subslot delivers the state filed in ChannelErrorType

Tab. 5-161: Enumeration type PNIO_DIAGTYPE

5.8.13 PNIO_LIVELIST

Brief description The data structure contains information about a Profinet IO device.

Structural element	Type	Description
IPAddress	STRING(16)	IP address of the Profinet IO device
SubNetMask	STRING(16)	Configuration of the Profinet IO device
Gateway	STRING(16)	Gateway address of the Profinet IO device
DeviceId	UINT	Device ID of the Profinet IO device
VendorId	UINT	Vendor ID of the Profinet IO devices

Structural element	Type	Description
MacAddress	ARRAY [0..5] OF BYTE	MAC address of the Profinet IO device
DeviceName	STRING(240)	Station name of the Profinet IO device
DeviceTypeName	STRING(240)	TypName of the Profinet IO device
DeviceRole	WORD	Role of the Profinet IO device, see below

Tab. 5-162: Data structure PNIO_LIVELIST

Structural element DeviceRole

Bit no.	Name	Description
0	PNIO device	The device implements the specified function
1	PNIO controller	
2	PNIO multidevice	
3	PNIO supervisor	
4...7	Reserved	- - -

The value of "DeviceRole" is coded as binary.

5.9 PROFINETIO_TABLE 16#200

5.9.1 General information

PROFINETIO_TABLE - ERROR_TABLE, table number 16#0200

Type definition

```
TYPE  ERROR_STRUCT :
    STRUCT
        Table       :ERROR_TABLE;
        Additional1 :DWORD;
        Additional2 :DWORD;
    END_STRUCT
END_TYPE
```

The table contains error numbers output by the following library function blocks:

- RIL_ProfiNetIO - Library for Profinet I/O controller (BusMaster),
- RIL_ProfiNetIODevice - Library for Profinet I/O devices (BusSlave).

The function blocks output the following error messages:

ErrorID	Additional1	Additional2	Note
INPUT_RANGE_ERROR	16#0001	xxx	Function block-dependent
COMMUNICATION_ERROR	**16#0003**	*Profinet I/O controller diagnostics* • General Error Codes , page 191, • Special Error Codes, Profinet I/O Controller, page 193..	
COMMUNICATION_ERROR	**16#0004**	*Profinet I/O status* • Special error codes, Profinet IO status , page 193..	
STATE_MACHINE_ERROR	16#0005	16#0000	No matching for response header
INPUT_INVALID_ERROR	16#0005	16#000x	IMxAdr is not set (x - I&M number)
INPUT_INVALID_ERROR	16#0006	16#0000	Invalid RW mode

ErrorID	Additional1	Additional2	Note
DEVICE_ERROR	16#0007	16#0000	Invalid I&M version (device I&M version has to be 1.1)
SYSTEM_ERROR	16#0101	Internal use	Controller not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	xxx	Error in state machine
INPUT_RANGE_ERROR	16#0106	0	Device 'IdentStr' is not configured
COMMUNICATION_ERROR	**16#3000**	*Profinet I/O I/O link master (IOL-M-error)* • Special error codes, IO I/O link master , page 203	
COMMUNICATION_ERROR	**16#3001**	*Profinet I/O I/O link device (application-related errors)* • Special error codes, Profinet I/O I/O link device (application-related errors) , page 203	
		Profinet I/O backup/restore	
INPUT_INVALID_ERROR	16#3002	16#0000	"BlockMode" input is invalid
INPUT_INVALID_ERROR	16#3002	16#0001	"ConfigFileName" is invalid
INPUT_INVALID_ERROR	16#3002	16#0002	"BackupFileName" is invalid
ACCESS_ERROR	16#3003	16#0000	"Configuration file" could not be read
ACCESS_ERROR	16#3003	16#0001	"Configuration file" could not be opened
ACCESS_ERROR	16#3003	16#0002	"Configuration file" / "backup file" format is invalid
ACCESS_ERROR	16#3003	16#0003	"Backup file" could not be opened
ACCESS_ERROR	16#3003	16#0004	Reading/writing "backup file" not possible

Tab. 5-163: Error codes Profinet I/O (controller), selection

5.9.2 General Error Codes

The error codes apply to all BusMasters and BusSlaves. They do not depend on the field bus.

The error codes are reported as **ERROR_STRUCT, Additional2**

for **ERROR_STRUCT, Additional1= 16#0003**.

Additional2	
16#800A0001	Invalid pointer (NULL) applied to driver
16#800A0002	No card available with the specified name/index
16#800A0003	No channel available with the specified index
16#800A0004	Invalid handle applied to driver
16#800A0005	Invalid parameters
16#800A0006	Invalid command
16#800A0007	Invalid buffer size
16#800A0008	Invalid access size
16#800A0009	Function failed
16#800A000A	File Could Not be Opened

Additional2	
16#800A000B	File size is zero
16#800A000C	Insufficient memory capacity to load the file
16#800A000D	Comparing file checksums failed
16#800A000E	Error when reading the file
16#800A000F	Invalid file type
16#800A0010	Invalid file name
16#800A0011	Driver function not available
16#800A0012	Specified buffer is too small
16#800A0013	Memory cannot be mapped
16#800A0014	No further entries present
16#800B0001	Driver not initialized
16#800B0002	Init status error of the driver
16#800B0003	Status read error of the driver
16#800B0004	Command is active on device
16#800B0005	General error upon download
16#800B0006	Wrong driver version
16#800B0030	CIFx driver not in operation
16#800B0031	Device could not be initialized
16#800B0032	Channel could not be initialized (xOpenChannel was not called)
16#800B0033	IOControl call failed
16#800B0034	Driver could not be opened
16#800C0010	The dual port memory could not be accessed (memory card not found)
16#800C0011	Device not ready (Ready flag failed)
16#800C0012	Device not ready for operation (Running flag failed)
16#800C0013	Watchdog test error
16#800C0015	Handshake flag error
16#800C0016	Mailbox for sent mails full
16#800C0017	Timeout for sending data packages
16#800C0018	Timeout for receiving data packages
16#800C0019	No data package available
16#800C001A	Insufficient memory space in mailbox
16#800C0020	Reset command timeout
16#800C0021	COM flag not set
16#800C0022	I/O data exchange failed
16#800C0023	I/O data exchange timeout
16#800C0024	Unknown I/O data exchange mode

Additional2	
16#800C0025	Device function failed
16#800C0026	DPM size “size” differs from the configuration
16#800C0027	Unknown status mode
16#800C0028	Output port is already used
16#800C0029	Timeout configuration lock
16#800C002A	Configuration unlock timeout
16#800C002B	Set HOST status timeout
16#800C002C	Delete HOST status timeout
16#800C002D	Timeout during channel initialization
16#800C002E	Set bus ON timeout
16#800C002F	Set bus OFF timeout

Tab. 5-164: General error codes

5.9.3 Special Error Codes, Profinet I/O Controller

The error codes shown apply to Profinet I/O controllers.

The error codes are reported as **ERROR_STRUCT, Additional2** for **ERROR_STRUCT, Additional1= 16#0003**.

Additional2	
16#C000001A	Acyclic channel is already assigned Only an acyclic service (DCP services or Read-Implicit service) can be executed, i.e. a new acyclic request can only be started after the previously started request has been completed. If a device cannot be reached, the acyclic channel is only available again after 7s (internal timeout). Affected function blocks: All function blocks with acyclic communication, backup&restore, device replacement, I/O link, IM data as well as the IL_PNIORemoteDeviceStateDetails function block.
16#C0000142	Connection timeout
16#C0000145	No Ethernet cable plugged in.
16#C0000180	The field bus is OFF.
16#C00C0038	The requested I/O device did not respond to the DCP request. It is possible that the device does not exist.
16#C00C0056	Service not available. The device is not in the I/O data exchange.
16#C00C0060	The acyclic service is acknowledged negatively by the device.
16#C00C0061	The acyclic service failed. The RPC layer detected an error in the received package.
16#C00C0062	The acyclic service failed. An internal error occurred.
16#C0140041	Invalid characters in the device name

Tab. 5-165: Special error codes of the Profinet I/O controller

5.9.4 Special error codes, Profinet IO status

The described error codes are the special error codes of the Profinet IO status.

These codings refer to the IEC 61158-6-10 Kap 6.2.5, Coding section related to PNIOStatus.

The error codes are reported as **ERROR_STRUCT, Additional2** at **ERROR_STRUCT, Additional1= 16#0004**.

Coding of Profinet IO status

Additional2-Maske	Meaning	
16#FF000000	**ErrorCode**	Error codes for negative responses
16#00FF0000	**ErrorDecode**	Selection of further decoding
16#0000FF00	**ErrorCode1**	Error codes 1
16#000000FF	**ErrorCode2**	Error codes 2

Example

The function block IL_PNIOReadRecord reports error ErrorIdent.Additional1 = 16#00000004 (≙ Profinet IO Status) and ErrorIdent.Additional2 = 16#DE80B000. This Additional2 error code is evaluated as follows:

- 16#**DE**xxxxxx : Error during read access, see "ErrorCode" on page 194
- 16#xx**80**xxxx : Error at read/write service "PNIORW", see "ErrorDecode" on page 195
- 16#xxxx**B0**xx : Bit 4-7 = 11 (decimal) → Bit 0-3 = 0 (decimal) → "invalid index", see "ErrorCode1/2 for ErrorCode PNIORW" on page 195, that means the index during read access was not acceptedby the PN-Device; it must be checked, if input "Index" is set correctly at the function block
- 16#xxxxxx**00** : ErrorCode2 has no meaning in this example

ErrorCode

Specification of the values for **ErrorCode**. Further values are defined, which will not occur in this context.

ErrorCode	Meaning	
16#20 - 16#3F	Manufacturer specific	Defined by manufacturer of the device
16#81	PNIO	Used for all errors not covered by the other domains
16#CF	RTA error	Used in RTA error PDUs. Used in alarm error telegrams (Low Level)
16#DA	AlarmAck	Used in RTA data PDUs. Used in alarm data telegrams (High Level)
16#DB	IODConnectRes	RPC Connect Response used to indicate errors occurred when handling a Connect request
16#DC	IODReleaseRes	RPC Release Response used to indicate errors occurred when handling a Release request
16#DD	IODControlRes	RPC ControlResponse used to indicate errors occurred when handling a Control request

ErrorCode	Meaning	
16#DE	IODReadRes	RPC Read Response used to indicate errors occurred when handling a Read request
16#DF	IODWriteRes	RPC Write Response Used to indicate error occurred when handling a Write response

ErrorDecode

Specification of the values for **ErrorDecode**. Further values are defined, which will not occur in this context.

ErrorDecode	Meaning	
16#80	PNIORW	For Read and Write services, see "ErrorCode1/2 for ErrorCode PNIORW" on page 195
16#81	PNIO	For other services, see "ErrorCode1/2 for ErrorDecode PNIO" on page 196
16#82	Manufacturer specific	Defined by manufacturer of the device

ErrorCode1/2 for ErrorCode "PNIORW"

Meaning ErrorCode1 for ErrorDecode "PNIORW"

ErrorClass (decimal) Bit7 – 4	Meaning	ErrorCode (decimal) Bit3 – 0
0 to 9 (0x0 to 0x9)	Not specified	Reserved
10 (0xA)	Application	0 = read error 1 = write error 2 = module failure 3,4,5,6 = not specified 7 = busy 8 = version conflict 9 = feature not supported 10 = user specific 1 …. 15 = user specific 6

ErrorClass (decimal) Bit7 – 4	Meaning	ErrorCode (decimal) Bit3 – 0
11 (0xB)	Access	0 = invalid index 1 = write length error 2 = invalid slot / subslot 3 = type conflict 4 = invalid area / API 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10= backup 11 = user specific 7 … 15 = user specific 11
12 (0xC)	Ressource	0 = read constrain conflict 1 = write constrain conflict 2 = resource busy 3 = resource unavailable 4,5,6,7 = not specified 8 = user specific 12 …. 15 = user specific 19
13 to 15 (0xD to 0xF)	User specific	User specific

Tab. 5-166: Specification of the coding of PNIOStatus.ErrorCode

The meaning of ErrorCode2 in this case is user-specific and is not executed further.

ErrorCode1/2 for ErrorDecode "PNIO"

Meaning ErrorCode1/2 for ErrorDecode "PNIO"

Error-Code1	Description	Error-Code2	Description
16#01	Connect Parameter Error. Faulty ARBlockReq	16#00 - 16#0D	Error in one of the parameter blocks
16#02	Connect Parameter Error. Faulty IOCRBlockReq	16#00 - 16#1C	Error in one of the parameter blocks
16#03	Connect Parameter Error. Faulty ExpectedSubmoduleBlockReq	16#00 - 16#10	Error in one of the parameter blocks
16#04	Connect Parameter Error. Faulty AlarmCRBlockReq	16#00 - 16#0F	Error in one of the parameter blocks

16#05	Connect Parameter Error. Faulty PrmServerBlockReq	16#00 - 16#08	Error in one of the parameter blocks
16#06	Connect Parameter Error. Faulty MCRBlockReq	16#00 - 16#08	Error in one of the parameter blocks
16#07	Connect Parameter Error. Faulty ARRPCBlockReq	16#00 - 16#04	Error in one of the parameter blocks
16#08	Read Write Record Parameter Error Faulty Record	16#00 - 16#0C	Error in one of the parameter blocks
16#09	Connect Parameter Error Faulty IRInfoBlock	16#00 - 16#05	Error in one of the parameter blocks
16#0A	Connect Parameter Error Faulty SRInfoBlock	16#00 - 16#05	Error in one of the parameter blocks
16#0B	Connect Parameter Error Faulty ARFSUBlock	16#00 - 16#05	Error in one of the parameter blocks
16#14	IODControl Parameter Error. Faulty ControlBlockConnect	16#00 - 16#09	Error in one of the parameter blocks
16#15	IODControl Parameter Error. FaultyControlBlockPlug	16#00 - 16#09	Error in one of the parameter blocks
16#16	IOXControl Parameter Error. ControlBlock after a connection establishment	16#00 - 16#07	Error in one of the parameter blocks
16#17	IOXControl Parameter Error. Faulty ControlBlock a plug alarm	16#00 - 16#07	Error in one of the parameter blocks
16#18	IODControl Parameter Error. Faulty ControlBlockPrmBegin	16#00 - 16#09	Error in one of the parameter blocks
16#19	IODControl Parameter Error Faulty SubmoduleListBlock	16#00 - 16#07	Error in one of the parameter blocks
16#28	Release Parameter Error. Faulty ReleaseBlock	16#00 - 16#07	Error in one of the parameter blocks
16#32	Response Parameter Error. Faulty ARBlockRes	16#00 - 16#08	Error in one of the parameter blocks
16#33	Response Parameter Error. Faulty IOCRBlockRes	16#00 16#06	Error in one of the parameter blocks

16#34	Response Parameter Error. Faulty AlarmCRBlockRes	16#00 - 16#06	Error in one of the parameter blocks
16#35	Response Parameter Error. Faulty ModuleDiffBlock	16#00 - 16#0D	Error in one of the parameter blocks
16#36	Response Parameter Error. Faulty ARRPCBlockRe	16#00 - 16#04	Error in one of the parameter blocks
16#37	Response Parameter Error Faulty ARServerBlockRes	16#00 - 16#05	Error in one of the parameter blocks
16#3C	AlarmAck Error Codes	16#00 16#01	Alarm type not supported Wrong submodule state
16#3D	CMDEV	16#00 16#01 16#02 - 16#FF	State conflict Resource State machine specific
16#3E	CMCTL	16#00 16#01 16#02	State conflict Timeout No data send
16#3F	CTLDINA	16#00 16#01 16#02 16#03 16#04 16#05 16#06	No DCP active DNS Unknown_RealStationName DCP No_RealStationName DCP Multiple_RealStationName DCP No_StationName No_IP_Addr DCP_Set_Error
16#40	CMRPC	16#00 16#01 16#02 16#03 16#04 16#05 16#06 16#07 16#08 16#09 16#0A 16#0B	ArgsLength invalid Unknown blocks IOCR mssing Wrong AlarmCRBlock count Out of AR resources AR UUID unknown State conflict Out of provider, consumer or AlarmResources Out of memory PDev already owned ARset State conflict during connection establishment ARset Parameter conflict during connection establishment

16#41	ALPMI	16#00 16#01	Invalid state Wrong ACK-PDU
16#42	ALPMR	16#00 16#01	Invalid state Wrong notification PDU
16#43	LMPM	16#00 - 16#FF	Ethernet switch errors
16#44	MAC	16#00 - 16#FF	-
16#45	RPC	16#01	CLRPC_ERR_REJECTED: EPM or server rejected the call
		16#02	CLRPC_ERR_FAULTED: Server had fault while executing the call
		16#03	CLRPC_ERR_TIMEOUT: EPM or server did not respond
		16#04	CLRPC_ERR_IN_ARGS; broadcast or maybe "ndr_data" too large
		16#05	CLRPC_ERR_OUT_ARGS: Server sent back more than "alloc_len"
		16#06	CLRPC_ERR_DECODE: Result of EPM lookup could not be decoded
		16#07	CLRPC_ERR_PNIO_OUT_ ARGS: Out-args not "PN IO signature", too short or inconsistent
		16#08	CLRPC_ERR_PNIO_APP_T IMEOUT: RPC call was terminated after RPC application timeout
16#46	APMR	16#00 16#01	Invalid state LMPM signaled an error
16#47	APMS	16#00 16#01 16#02	Invalid state LMPM signaled an error Timeout
16#48	CPM	16#00	Invalid state
16#49	PPM	16#00	Invalid state
16#4A	DCPUCS	16#00 16#01 16#02	Invalid state LMPM signaled an error Timeout

16#4B	DCPUCR	16#00 16#01	Invalid state LMPM signaled an error
16#4C	DCPMCS	16#00 16#01	Invalid state LMPM signaled an error
16#4D	DCPMCR	16#00 16#01	Invalid state LMPM signaled an error
16#4E	FSPM	16#00 - 16#FF	FAL service protocol machine error
16#64	CTLSM	16#00 16#01	Invalid state CTLSM signaled an error
16#65	CTLRDI	16#00 16#01	Invalid state CTLRDI signaled an error
16#66	CTLRDR	16#00 16#01	Invalid state CTLRDR signaled an error
16#67	CTLWRI	16#00 16#01	Invalid state CTLWRI signaled an error
16#68	CTLWRR	16#00 16#01	Invalid state CTLWRR signaled an error
16#69	CTLIO	16#00 16#01	Invalid state CTLIO signaled an error
16#6A	CTLSU	16#00 16#01 16#02 16#03 16#04 16#05	Invalid state AR add provider or consumer failed AR alarm-open failed AR alarm-ack-send AR alarm-send AR alarm-ind
16#6B	CTLRPC	16#00 16#01	Invalid state CTLRPC signaled an error
16#6C	CTLPBE	16#00 16#01	Invalid state CTLPBE signaled an error
16#C8	CMSM	16#00 16#01	Invalid state CMSM signaled an error
16#CA	CMRDR	16#00 16#01	Invalid state CMRDR signaled an error
16#CC	CMWRR	16#00 16#01 16#02	Invalid state AR is not in state Primary (write not allowed) CMWRR signaled an error

16#CD	CMIO	16#00 16#01	Invalid state CMIO signaled an error
16#CE	CMSU	16#00 16#01 16#02 16#03 16#04 16#05	Invalid state AR add provider or consumer failed AR alarm-open failed AR alarm-send AR alam-ack-send AR alarm-ind
16#D0	CMINA	16#00 16#01	Invalid state CMINA signaled an error
16#D1	CMPBE	16#00 16#01	Invalid state CMPBE signaled an error
16#D2	CMDMC	16#00 16#01	Invalid state CMDMC signaled an error

16#FD	Used by RTA for protocol error (RTA_ERR_CLS_PROTO-COL)	16#01	Error within the coordination of sequence numbers (RTA_ERR_CODE_SEQ)
		16#02	Instance closed (RTA_ERR_ABORT)
		16#03	AR out of memory (RTA_ERR_ABORT)
		16#04	AR add provider or consum-er failed (RTA_ERR_ABORT)
		16#05	AR consumer DHT / WDT expired (RTA_ERR_ABORT)
		16#06	AR cmi timeout (RTA_ERR_ABORT)
		16#07	AR alarm-open failed (RTA_ERR_ABORT)
		16#08	AR alarm-send.cnf(-) (RTA_ERR_ABORT)
		16#09	AR alarm-ack- send.cnf(-) (RTA_ERR_ABORT)
		16#0A	AR alarm data too long (RTA_ERR_ABORT)
		16#0B	AR alarm.ind(err) (RTA_ERR_ABORT)
		16#0C	AR rpc-client call.cnf(-) (RTA_ERR_ABORT)
		16#0D	AR abort.req (RTA_ERR_ABORT)
		16#0E	AR re-run aborts existing (RTA_ERR_ABORT)
		16#0F	AR release.ind received (RTA_ERR_ABORT)
		16#10	AR device deactivated (RTA_ERR_ABORT)
		16#11	AR removed (RTA_ERR_ABORT)
		16#12	AR protocol violation (RTA_ERR_ABORT)
		16#13	AR name resolution error (RTA_ERR_ABORT)
		16#14	AR RPC-Bind error (RTA_ERR_ABORT)
		16#15	AR RPC-Connect error (RTA_ERR_ABORT)
		16#16	AR RPC-Read error (RTA_ERR_ABORT)
		16#17	AR RPC-Write error (RTA_ERR_ABORT)
		16#18	AR RPC-Control error (RTA_ERR_ABORT)
		16#19	AR forbidden pull or plug af-

16#FF	User specific	16#00 - 16#FE 16#FF	User specific Recommended for “User abort” without further detail

Tab. 5-167: Specification of the coding of PNIOStatus.ErrorCode

5.9.5 Special error codes, IO I/O link master

The shown error codes apply to the I/O link master.

The error codes are reported as **ERROR_STRUCT, Additional2** for **ERROR_STRUCT, Additional1= 16#3000**.

Special errors of the I/O link master

ERROR_CODE	Additional2	
COMMUNICATION_ERROR	16#7000	IOL_CALL conflict. Unexpected Write request instead of Read request
COMMUNICATION_ERROR	16#7001	Wrong IOL_CALL. Decode error
COMMUNICATION_ERROR	16#7002	Port blocked. Port occupied by another task
COMMUNICATION_ERROR	16#8000	Timeout when IOL-Ds or IOL-M ports are busy
COMMUNICATION_ERROR	16#8001	Wrong index. IOL_Index 32767 ... 65535
COMMUNICATION_ERROR	16#8002	Wrong port address. Port address beyond defined maximum
COMMUNICATION_ERROR	16#8003	Wrong port function. Port function not supported

Tab. 5-168: Error codes of the PNIO_IO link master

5.9.6 Special error codes, Profinet I/O I/O link device (application-related errors)

The error codes shown apply to the Profinet IO I/O link device.

The error codes are reported as **ERROR_STRUCT, Additional2** for **ERROR_STRUCT, Additional1= 16#3001**.

Special errors of the Profinet I/O I/O link device (application-related errors)

ERROR_CODE	Additional2	
COMMUNICATION_ERROR	16#8000	Device application error – no details
COMMUNICATION_ERROR	16#8011	Index not available
COMMUNICATION_ERROR	16#8012	SubIndex not available
COMMUNICATION_ERROR	16#8020	Service temporarily not available
COMMUNICATION_ERROR	16#8021	Service temporarily not available – Local control
COMMUNICATION_ERROR	16#8022	Service temporarily not available – Device control
COMMUNICATION_ERROR	16#8023	Access denied
COMMUNICATION_ERROR	16#8030	Parameter value out of range
COMMUNICATION_ERROR	16#8031	Parameter value above limit
COMMUNICATION_ERROR	16#8032	Parameter value below limit
COMMUNICATION_ERROR	16#8033	Parameter length overrun
COMMUNICATION_ERROR	16#8034	Parameter length under run

ERROR_CODE	Additional2	
COMMUNICATION_ERROR	16#8035	Function not available
COMMUNICATION_ERROR	16#8036	Function temporarily unavailable
COMMUNICATION_ERROR	16#8040	Invalid parameter set
COMMUNICATION_ERROR	16#8041	Inconsistent parameter set
COMMUNICATION_ERROR	16#8082	Application not ready
COMMUNICATION_ERROR	16#8001..16#81FF	Vendor-specific

Tab. 5-169: Error codes, Profinet IO I/O link device

5.10 Example programs

Example programs to diagnose the Profinet I/O controller are exemplarily provided as GAT plugin, e.g. the example program "PNIODiagLogger" can also be used with the control without GAT.

The GAT plug-ins are installed with IndraWorks (with 'x' = Number of IndraWorks installation):

```
C:\ProgramData\Rexroth\IndraWorks\'x'\GatWizard\Plugins\
```

GAT plugins consist of:

- an *.iwx file that can be imported below an application
- a brief documentation (*.pdf, mainly in English)
- a configuration file *Config.txt, only required by the GAT Wizard

For further information, refer to the “IndraLogic XLC IndraMotion MLC 14VRS, Generic Application Template” (DOK-XLCMLC-TF*GAT**V14-APRS-EN-P).

6 RIL_ProfinetIODevice.library

6.1 Introduction and overview

The controls can be operated as field bus slaves. In case of Profinet I/O, the field bus slave is called Profinet I/O device. The library describes the interfaces of this Profinet I/O device to the IEC user program.

Implementation of the library is based on RIL_UTILITIES. The IL_EnableDoneBase and IL_ExecuteDoneBase methods of the basic function blocks (basic classes) are used and extended or overwritten.

Areas of application and function blocks of RIL_ProfinetIODevice

Area of application (folder in library)	
Function blocks	**Description**
Diagnostics	
IL_PNIODeviceState, page 205	to determine the status of a Profinet device
IL_PNIODeviceStateDetails, page 207 *using data type* • IL_PNIO_DEVICE_STATE, page 209	to determine the status of the Profinet device stack
IL_PNIODeviceStateDetailsXMAC, page 208 *using data types* • IL_PNIO_DEVICE_XMAC, page 211 • IL_PNIO_DEVICE_XMAC_PORT, page 211	status of the Profinet 2 port switch

Tab. 6-1: RIL_ProfinetIODevice library

Addressing the device

Addressing: The Profinet I/O devices are distinguished based on their ascending order in the configuration.

Counting starts with 0th device, i.e., if there is only one configured device, its device instance is always 0.

Also refer to IL_BUSSLAVE, page 212.

PROFINETIO_TABLE 16#200

The PROFINETIO_TABLE 16#200, page 213, contains error numbers output by the function blocks of the library.

6.2 Diagnostics (POUs/Diagnosis)

6.2.1 IL_PNIODeviceState

Brief description

This function block returns a basic diagnosis of the PROFINET IO device. If a diagnostic message is present, details on the diagnoses can be read via additional FBs (e.g., IL_PNIODeviceStateDetails).

Target system	Library/folder
MLC	RIL_ProfinetIODevice/POUs/Diagnosis
MTX	RIL_ProfinetIODevice/POUs/Diagnosis

Tab. 6-2: Reference table

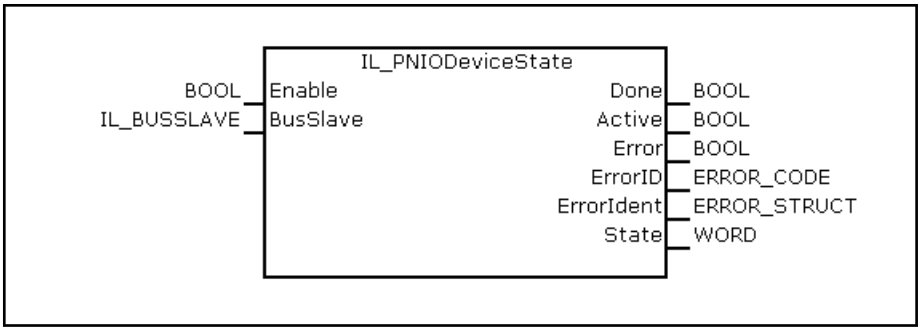


Fig. 6-1: IL_PNIODeviceState

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the field bus slave in the order of the configuration tree. Also refer to IL_BUSSLAVE on page 212
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	State	WORD	State information on the local field bus slave

Tab. 6-3: Function block IL_PNIODeviceState

The "State" output provides a rough diagnostic classification in binary code.

The bits are contained in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Output "State"

Bit no.	Name	Description
0	IL_FBS_BUS_OFF	No field bus communication
2	IL_FBS_CONFIG_ERROR	Faulty configuration by the field bus master
4	IL_FBS_SLAVE_ERROR	Device error
5	IL_FBS_UNUSED_MODULE_INFO	Information: Unused IO module; the IO modules are not operated by the controller.

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Enable

Tab. 6-4: Minimum / maximum values and default values of the IL_PNIODeviceState inputs

Functional description

The function block retrieves state information about the PROFINET device and returns it as bit array in "State". The function block is completely processed in one PLC cycle.

Troubleshooting The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200). It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	Internal use	Device not found in configuration table

Tab. 6-5: Error codes of the IL_PNIODeviceState function block

6.2.2 IL_PNIODeviceStateDetails

Brief description This function block determines the detailed status of the PROFINET device stack. It provides information on

- the state of the communication stack,
- the state of the bus communication
- the previous error values.

Target system	Library/folder
MLC	RIL_ProfinetIODevice/ POUs/ Diagnosis
MTX	RIL_ProfinetIODevice/ POUs/ Diagnosis

Tab. 6-6: Reference table

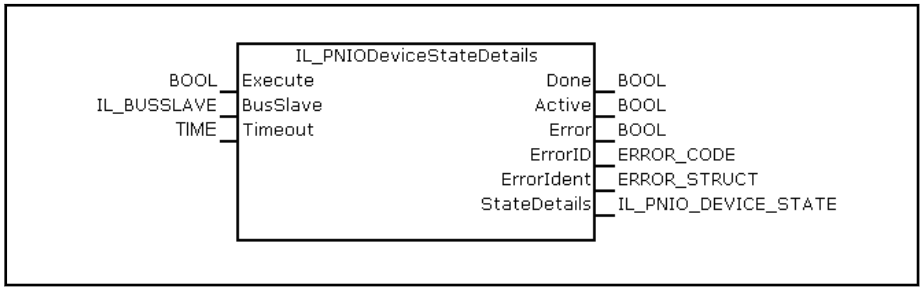


Fig. 6-2: IL_PNIODeviceStateDetails

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the field bus slave in the order of the configuration tree. Also refer to IL_BUSSLAVE on page 212
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	StateDetails	IL_PNIO_DEVICE_STATE	Detailed information about the status of the device stack IL_PNIO_DEVICE_STATE on page 209

Tab. 6-7: Function block IL_PNIODeviceStateDetails

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Execute
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at “Execute”

Tab. 6-8: Minimum / maximum values and default values of the IL_PNIODeviceStateDetails inputs

Functional description When activated, this function block determines the current state of the device once.

Troubleshooting The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200). It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	Internal use	Device not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
COMMUNICATION_ERROR	16#0003	*PROFINET I/O device diagnostics* • See General error codes, page 213. • See Special error codes, PROFINET I/O device, page 215.	

Tab. 6-9: Error codes of function block IL_PNIODeviceStateDetails

6.2.3 IL_PNIODeviceStateDetailsXMAC

Brief description This function block determines the detailed status of the Profinet Switch on the netX© communication controller. It provides information on each Ethernet port about

- its connectivity and
- the statistics counter of that port.

Target system	Library/folder
MLC	RIL_ProfinetIODevice/ POUs/ Diagnosis
MTX	RIL_ProfinetIODevice/ POUs/ Diagnosis

Tab. 6-10: Reference table

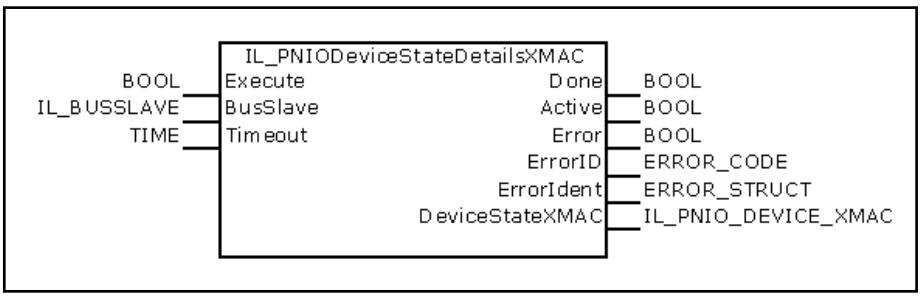


Fig. 6-3: IL_PNIOGetDeviceXMACState

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the field bus slave in the order of the configuration tree. IL_BUSSLAVE, page 212
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	DeviceStateXMAC	IL_PNIO_DEVICE_XMAC	Detailed state information about the two ports of the communication controller IL_PNIO_DEVICE_XMAC, page 211

Tab. 6-11: Function block IL_PNIOGetDeviceXMACState

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Execute
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at “Execute”

Tab. 6-12: Minimum / maximum values and default values of the IL_PNIODeviceStateDetailsXMAC inputs

Functional description When activated, the function block once determines the current statistical information on the two Ethernet ports of the communication controller.

Troubleshooting The function block uses the error table **PROFINETIO_TABLE** (ERROR_TABLE = 16#0200). It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	Internal use	Device not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in state machine
COMMUNICATION_ERROR	16#0003	*PROFINET I/O device diagnostics* • See General error codes, page 213. • See Special error codes, PROFINET I/O device, page 215.	

Tab. 6-13: Error codes of function block IL_PNIODeviceStateDetailsXMAC

6.3 Data types (DataTypes)

6.3.1 IL_PNIO_DEVICE_STATE

Brief description Data structure IL_PNIO_DEVICE_STATE contains the state information on the PNIO device stack.

Structural element	Type	Description
PnsState	UDINT	State of the PROFINET IO device task
LastRslt	UDINT	Previous error code. The meaning of the error can be learned from the error table.
LinkState	UDINT	Link state of the network connection
ConfigState	UDINT	Configuration status
CommunicationState	UDINT	Network status of the communication channel
CommunicationError	UDINT	Current error code of the communication channel. The meaning of the error can be learned from the error table.

Tab. 6-14: Data structure IL_PNIO_DEVICE_STATE

The structure elements can have the following values:

PnsState

The 'PnsState' element of data structure IL_PNIO_DEVICE_STATE describes the status of the PNIO device via bit code. Multiple bits can be set at the same time.

Bit	Value	
7	16#00000080	Cyclic data exchange of the slot
6	16#00000040	Enabled for cyclic data exchange
5	16#00000020	Module 0 configured at submodule1
4	16#00000010	Module 0 is configured
3	16#00000008	At least one API (Application ProcessIndentifier) is created
2	16#00000004	Reserved
1	16#00000002	The Profinet stack is started
0	16#00000001	Device information is set

LinkState

The 'LinkState' element of data structure IL_PNIO_DEVICE_STATE describes the status of the network connection.

Value	
0	No information available
1	No physical connection established
2	Slow (10 Mbit) velocity
3	No physical connection present

ConfigState

Value	
0	Not configured
1	Configured via database
2	Configuration database faulty
3	Configured by application
4	Configuration by application
5	Error upon configuration by application

Value	
6	Configuration with warm start parameters
7	Configuration via warm start parameters
8	Error when configuring with warm start parameters

CommunicationState The 'CommunicationState' element of data structure IL_PNIO_DEVICE_STATE describes the status of communication of the PNIO device.

Value	Name	
0	UNKNOWN	Communication status unknown
1	OFFLINE	No PN communication
2	STOP	No PN communication possible or allowed
3	IDLE	Reserved
4	OPERATE	PN communication allowed (conclusions about cyclic communication cannot be drawn)

6.3.2 IL_PNIO_DEVICE_XMAC

Brief description The structure contains the status and statistic information on the subordinate Profinet 2-port switch.

A structure of type IL_PNIO_DEVICE_XMAC_PORT, page 211 is stored for each available port.

Type	Name	Description
ARRAY [0..1] of IL_PNIO_DEVICE_XMAC_PORT	XmacPort[]	State of the Profinet IO device task

Tab. 6-15: Data structure IL_PNIO_DEVICE_XMAC

Assignment of pin names to the ports with indexes 0 and 1 in field aXmacDiag[]:

Control	Port 0	Port 1
CFL01_1_TP	X7E1	X7E2

6.3.3 IL_PNIO_DEVICE_XMAC_PORT

Brief description The structure contains the status and statistic information on a switch port.

Name	Type	
FramesTransmittedOk	UDINT	Number of successfully sent telegrams
SingleCollisionFrames	UDINT	Number of telegrams with one single collision
MultipleCollisionFrames	UDINT	Number of telegrams with multiple collisions
LateCollisions	UDINT	Number of “late collisions” (after more than 512 bit times)
LinkDownDuringTransmission	UDINT	Number of detected telegrams when exiting the connection
UtxUnderflowDuringTransmission	UDINT	Number of FIFO idle states while sending
TxFatalErrors	UDINT	Number of fatal sending errors

Name	Type	
FramesReceivedOk	UDINT	Number of successfully received telegrams
FrameCheckSequenceErrors	UDINT	Number of correctly framed telegrams with a block check sum error
AlignmentErrors	UDINT	Number of incorrectly framed telegrams with a block check sum error
FrameTooLongErrors	UDINT	Number of telegrams received is higher than the telegram sized permitted
RuntFramesReceived	UDINT	Number of telegrams between 42 and 63 bytes with a valid checksum
CollisionFragmentsReceived	UDINT	Number of telegrams below 64 bytes with a checksum error
FramesDroppedDueLowResource	UDINT	Number of telegrams discarded due to lack of resources
FramesDroppedDueUrxOverflow	UDINT	Number of telegrams discarded due to FIFO overflow
RxFatalErrors	UDINT	Number of fatal reception errors

Tab. 6-16: IL_PNIO_DEVICE_XMAC_PORT structure

6.4 IL_BUSSLAVE

Brief description

The IL_BUSSLAVE enumeration type of the RIL_FieldbusTypes.library serves to select the appropriate field bus slave. In addition to the onboard real-time Ethernet interface, other devices can also be available in an MLC system via function modules.

The devices are distinguished based on their ascending order in the configuration. Indexes are assigned to the devices in the following order

- Onboard
- Function module 1
- Function module 2
- Function module 3
- Function module 4

Element	Value	Description
IL_BUSSLAVE_0	0	First device
IL_BUSSLAVE_1	1	Second device
IL_BUSSLAVE_2	2	Third device
IL_BUSSLAVE_3	3	Fourth device
IL_BUSSLAVE_4	4	Fifth device

Tab. 6-17: Elements of the IL_BUSSLAVE enumeration type

Examples of addressing the various instances of the PROFINET IO devices on a control:

Configured:	Addressed with:
Onboard: PROFINET IO device	IL_BUSSLAVE_0
FM1: Profinet I/O controller	IL_BUS**MASTER**_0
FM2: PROFINET IO device	IL_BUSSLAVE_1
FM3: PROFINET IO device	IL_BUSSLAVE_2

Tab. 6-18: Enumeration type IL_BUSSLAVE, example

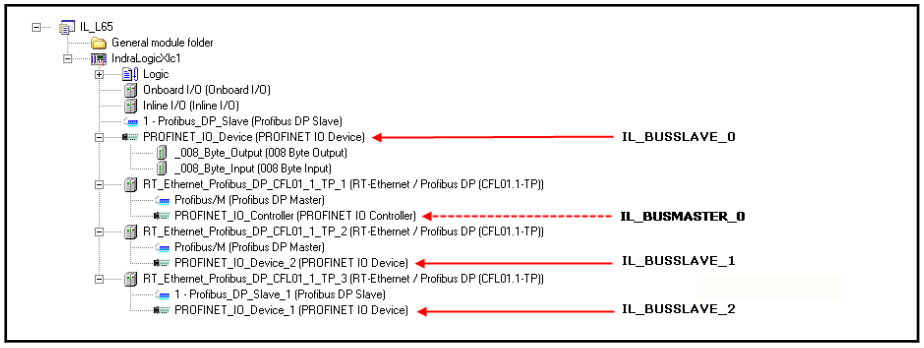


Fig. 6-4: Enumeration type IL_BUSSLAVE, example

6.5 PROFINETIO_TABLE 16#200

6.5.1 General information

PROFINETIO_TABLE - ERROR_TABLE, table number 16#0200

Type definition

```
TYPE  ERROR_STRUCT :
    STRUCT
        Table       :ERROR_TABLE;
        Additional1 :DWORD;
        Additional2 :DWORD;
    END_STRUCT
END_TYPE
```

The table contains error numbers output by the following library function blocks:

- RIL_ProfiNetIO - Library for PROFINET I/O controller (BusMaster),
- RIL_ProfiNetIODevice - Library for PROFINET I/O devices (BusSlave).

The function blocks output the following error messages:

ErrorID	Additional1	Additional2	
COMMUNICATION_ERROR	16#0003	*PROFINET I/O device diagnostics* • See General error codes, page 213. • See Special error codes, PROFINET I/O device, page 215.	
SYSTEM_ERROR	16#0101	Internal use	Device not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	xxx	Error in state machine

Tab. 6-19: Error codes ProfiNet I/O (device), selection

6.5.2 General error codes

The error codes apply to all BusMasters and BusSlaves. They do not depend on the field bus.

The error codes are reported as **ERROR_STRUCT, Additional2**

for **ERROR_STRUCT, Additional1= 16#0003**.

Additional2	
16#800A0001	Invalid pointer (NULL) applied to driver
16#800A0002	No card available with the specified name/index

Additional2	
16#800A0003	No channel available with the specified index
16#800A0004	Invalid handle applied to driver
16#800A0005	Invalid parameters
16#800A0006	Invalid command
16#800A0007	Invalid buffer size
16#800A0008	Invalid access size
16#800A0009	Function failed
16#800A000A	File Could Not be Opened
16#800A000B	File size is zero
16#800A000C	Insufficient memory capacity to load the file
16#800A000D	Comparing file checksums failed
16#800A000E	Error when reading the file
16#800A000F	Invalid file type
16#800A0010	Invalid file name
16#800A0011	Driver function not available
16#800A0012	Specified buffer is too small
16#800A0013	Memory cannot be mapped
16#800A0014	No further entries present
16#800B0001	Driver not initialized
16#800B0002	Init status error of the driver
16#800B0003	Status read error of the driver
16#800B0004	Command is active on device
16#800B0005	General error upon download
16#800B0006	Wrong driver version
16#800B0030	CIFx driver not in operation
16#800B0031	Device could not be initialized
16#800B0032	Channel could not be initialized (xOpenChannel was not called)
16#800B0033	IOControl call failed
16#800B0034	Driver could not be opened
16#800C0010	The dual port memory could not be accessed (memory card not found)
16#800C0011	Device not ready (Ready flag failed)
16#800C0012	Device not ready for operation (Running flag failed)
16#800C0013	Watchdog test error
16#800C0015	Handshake flag error
16#800C0016	Mailbox for sent mails full
16#800C0017	Timeout for sending data packages

Additional2	
16#800C0018	Timeout for receiving data packages
16#800C0019	No data package available
16#800C001A	Insufficient memory space in mailbox
16#800C0020	Reset command timeout
16#800C0021	COM flag not set
16#800C0022	I/O data exchange failed
16#800C0023	I/O data exchange timeout
16#800C0024	Unknown I/O data exchange mode
16#800C0025	Device function failed
16#800C0026	DPM size “size” differs from the configuration
16#800C0027	Unknown status mode
16#800C0028	Output port is already used
16#800C0029	Timeout configuration lock
16#800C002A	Configuration unlock timeout
16#800C002B	Set HOST status timeout
16#800C002C	Delete HOST status timeout
16#800C002D	Timeout during channel initialization
16#800C002E	Set bus ON timeout
16#800C002F	Set bus OFF timeout

Tab. 6-20: General error codes

6.5.3 Special error codes, PROFINET I/O device

The error codes shown apply to PROFINET I/O devices.

The error codes are reported as **ERROR_STRUCT, Additional2** for **ERROR_STRUCT, Additional1= 16#0003**.

Additional2	
16#C0300001	Invalid command
16#C0300004	Profinet I/O device setup missing

Tab. 6-21: Special error codes of a Profinet I/O device

7 RIL_EtherNetIP.library

7.1 Introduction and overview

The **RIL_EthernetIP.library** library is the interface from the PLC program to the Ethernet/IP devices. Ethernet/IP devices can be diagnosed via this interface.

Furthermore, function blocks, which can acyclically read and write individual data or complete data blocks, are described.

The IL_EnableDoneBase and IL_ExecuteDoneBase methods are used and extended or overwritten.

Target systems

The library can be used in the following systems:

Target system	Library
MLC	RIL_EtherNetIP
MTX	RIL_EtherNetIP

Tab. 7-1: Reference table

Areas of application and function blocks of the RIL_EthernetIP.library

Area of application (folder in library)	
Function blocks (POUs)	**Description**
Acyclic services (Acyclic_Communication)	
IL_ENIPGetAttributeSingle, page 218	Reads a "Class-Instance-Attribute" value
IL_ENIPGetIdentityObjAllAttributes, page 220	Reads all mandatory attributes of the "Identity object" of an adapter
IL_ENIPSendUnconnectedMessage, page 222	Can execute any acyclic CIP service
IL_ENIPSetAttributeSingle, page 224	Writes a "Class-Instance-Attribute" value
Backing up and restoring (Backup Restore)	
IL_ENIPAdapterBackup, page 226	Backs up the configuration and the parameters of an Ethernet/IP adapter device to a file via acyclic reading
IL_ENIPAdapterRestore, page 230	Restores parameters of an Ethernet/IP adapter device from the backup file via acyclic writing
Diagnostics (Diagnosis)	
IL_ENIPRemoteAdapterState, page 232	Reads the basic diagnostics of the Ethernet/IP adapter at the scanner
IL_ENIPRemoteAdapterStateDetails, page 233	Reads the detailed diagnostics of the Ethernet/IP adapter at the scanner
IL_ENIPScannerState, page 236	Reads the basic diagnostics of the Ethernet/IP scanner
IL_ENIPScannerStateDetails, page 237	Reads the detailed diagnostics of the Ethernet/IP scanner
With data types:	
IL_ENIP_IDENTITY_ATTRIBUTES, page 239	Contains the mandatory attributes of the "Identity object" read via "Get-Attribute-All"
IL_ENIP_REMOTE_ADAPTER_STATE_DETAILS, page 240	Allows to display the bits 4...7 "Detailed status" of an Ethernet/IP adapter (corresponds to attribute 5 of the "Identity object") as text

IL_ENIP_SCANNER_STATE, page 241	Structure, containing diagnostic information on the state of the communication stack, the state of the bus communication, the error counter as well as an overview on configured, active and slaves reporting diagnostics
With data types from RIL_FieldbusTypes	
IL_FBUS_COMMUNICATION_STATE, page 185	Status of the field bus communication
IL_FBUS_SLAVE_STATE, page 185	Status of the communication relations to one adapter (slave)

Tab. 7-2: RIL_EthernetIP library

Selecting the Ethernet/IP scanner

Addressing:

The Ethernet/IP scanners are distinguished based on their ascending order in the configuration.

Counting starts with "0". If there is only one configured scanner, its scanner instance is always "0" (also refer to IL_BUSMASTER, page 239).

Addressing the IP address of the Ethernet/IP adapter

IP address of the adapter to be accessed.

7.2 Acyclic services (Acyclic_Communication)

7.2.1 IL_ENIPGetAttributeSingle

Brief description

An individual attribute can be retrieved using the **IL_ENIPGetAttributeSingle** attribute.

Target system	Library/folder
MLC	RIL_EtherNetIP/POUs/Acyclic_Communication
MTX	RIL_EtherNetIP/POUs/Acyclic_Communication

Tab. 7-3: Reference table

Interface description

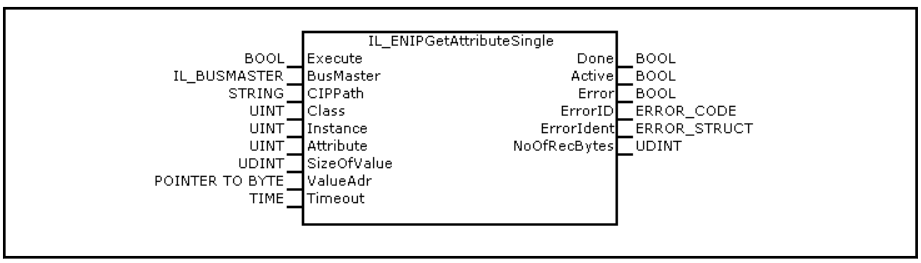


Fig. 7-1: Interface description of the IL_ENIPGetAttributeSingle function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Ethernet/IP scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string "aaa.bbb.ccc.ddd"
	Class	UINT	Class code
	Instance	UINT	Instance number
	Attributes	UINT	Attribute ID
	SizeOfValue	UDINT	Size of the data range addressed with "ValueAdr"
	ValueAdr	POINTER TO BYTE	Initial address of the data range
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NoOfRecBytes	UDINT	Number of bytes received in the data range

Tab. 7-4: I/O interface of the IL_ENIPGetAttributeSingle function block

The input variables can assume the values specified in the following table.

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
CIPPath	STRING	Character string	Character string	„ “	Rising edge at "Execute"
Class	UINT	16#01	16#FF	16#01	Rising edge at "Execute"
Instance	UINT	16#00	16#FF	16#00	Rising edge at "Execute"
Attributes	UINT	16#01	16#FF	16#01	Rising edge at "Execute"
SizeOfValue	UDINT	16#0000	16#01C2	16#0000	Rising edge at "Execute"
ValueAdr	POINTER TO BYTE	N. def.	(450 bytes)	16#0000	Rising edge at "Execute"
Timeout	TIME	T#0ms	N. def.	T#2s	Rising edge at "Execute"

Tab. 7-5: Min./max. and default values of the IL_ENIPGetAttributeSingle inputs

Functional description

An individual attribute can be retrieved using the **IL_ENIPGetAttributeSingle** attribute.

The device (adapter) is addressed via the IP address at first.

"Class", "Instance" and "Attribute" are used to access a certain "value" (or setting) of the adapter.

The attribute has a tree structure. However, class attributes are also available. In this case, the "Instance" is not specified or the "Instance" number is the attribute.

Addressing is manufacturer-specific. Usually, the following default objects are supported:

Identity Object	(Class Code: 0x01)
Message Router Object	(Class Code: 0x02)
Assembly Object	(Class Code: 0x04)
Connection Object	(Class Code: 0x05)
Connection Manager Object	(Class Code: 0x06)
TCP/IP Interface Object	(Class Code: 0xF5)
EtherNet Link Object	(Class Code: 0xF6)

For further information, refer to the manufacturer documentation and CIP specification

Troubleshooting

The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.2.2 IL_ENIPGetIdentityObjAllAttributes

Brief description

The **IL_ENIPGetIdentityObjAllAttributes** function block retrieves all mandatory attributes from the "Identity Object" of the adapter (CIP service "Get-Attribute-All" executed in background).

Target system	Library/folder
MLC for 14V06 and higher	RIL_EtherNetIP/POUs/Acyclic_Communication
MTX from 14V06	RIL_EtherNetIP/POUs/Acyclic_Communication

Tab. 7-6: Reference table

Interface description

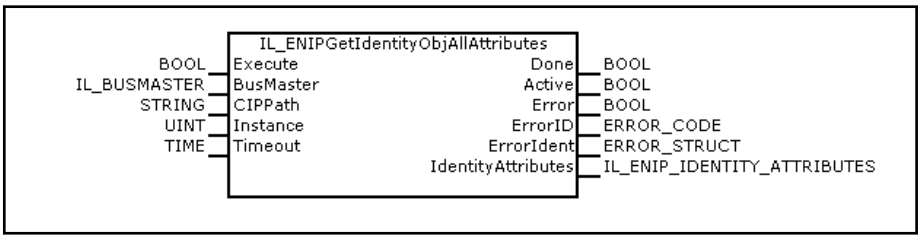


Fig. 7-2: Interface description of the IL_ENIPGetIdentityObjAllAttributes function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Ethernet/IP scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string "aaa.bbb.ccc.ddd"
	Instance	UINT	Instance number; generally, one adapter has only one "Identity Object" (instance = 1)
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	IdentityAttributes	IL_ENIP_IDENTITY_ATTRIBUTES	Mandatory attributes of the "Identity Object" Refer to chapter 7.5.2 "IL_ENIP_IDENTITY_ATTRIBUTES" on page 239

Tab. 7-7: I/O interface of the IL_ENIPGetIdentityObjAllAttributes function block

The input variables can assume the values specified in the following table.

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
CIPPath	STRING	Character string	Character string	„ “	Rising edge at "Execute"
Instance	UINT	0	255	1	Rising edge at "Execute"
Timeout	TIME	T#0ms	N. def.	T#2s	Rising edge at "Execute"

Tab. 7-8: Min./max. and default values of the IL_ENIPGetIdentityObjAllAttributes inputs

Functional description

The **IL_ENIPGetIdentityObjAllAttributes** function block retrieves all mandatory attributes from the "Identity Object" of the adapter (CIP service "Get-Attribute-All" executed in background).

The device (adapter) is addressed via the IP address at first.

Generally, one adapter has only one instance of the "Identity Object" (instance = 1).

The mandatory attributes are retrieved via "Get-Attribute-All" from the "Identity Object".

Troubleshooting The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.2.3 IL_ENIPSendUnconnectedMessage

Brief description Use the **IL_ENIPSendUnconnectedMessage** function block to execute any acyclic service. The function blocks chapter 7.2.1 "IL_ENIPGetAttributeSingle" on page 218 and chapter 7.2.4 "IL_ENIPSetAttributeSingle" on page 224 also use this function in the background for example.

Target system	Library/folder
MLC for 14V06 and higher	RIL_EtherNetIP/POUs/Acyclic_Communication
MTX from 14V06	RIL_EtherNetIP/POUs/Acyclic_Communication

Tab. 7-9: Reference table

Interface description

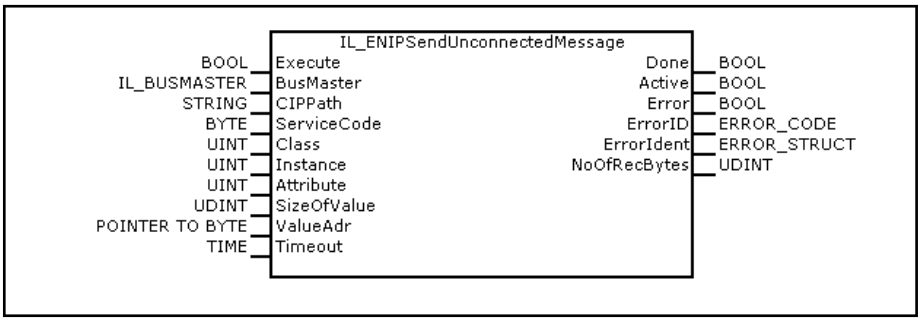


Fig. 7-3: Interface description of the IL_ENIPSendUnconnectedMessage function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Ethernet/IP scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string "aaa.bbb.ccc.ddd"
	Service Code	BYTE	CIP service code
	Class	UINT	Class code
	Instance	UINT	Instance number
	Attributes	UINT	Attribute ID
	SizeOfValue	UDINT	Size of the data range addressed with ValueAdr
	ValueAdr	POINTER TO BYTE	Initial address of the data range
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NoOfRecBytes	UDINT	Number of bytes received in the data range

Tab. 7-10: I/O interface of the IL_ENIPSendUnconnectedMessage function block

The input variables can assume the values specified in the following table.

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
CIPPath	STRING	Character string	Character string	„ “	Rising edge at "Execute"
Service Code	BYTE	1	255	0	Rising edge at "Execute"
Class	UINT	16#01	16#FF	16#01	Rising edge at "Execute"
Instance	UINT	16#00	16#FF	16#00	Rising edge at "Execute"
Attributes	UINT	16#01	16#FF	16#01	Rising edge at "Execute"
SizeOfValue	UDINT	16#0000	16#01C2	16#0000	Rising edge at "Execute"
ValueAdr	POINTER TO BYTE	N. def.	(450 bytes)	16#0000	Rising edge at "Execute"
Timeout	TIME	T#0ms	N. def.	T#2s	Rising edge at "Execute"

Tab. 7-11: Min./max. and default values of the IL_ENIPSendUnconnectedMessage inputs

Functional description

Use the **IL_ENIPSendUnconnectedMessage** function block to execute any CIP service. Either a standard CIP service code or a vendor- or device-specific CIP service can be executed.

The device (adapter) is addressed via the IP address at first.

"Class", "Instance" and "Attribute" are used to access a certain "value" (or setting) of the adapter. There are the following standard classes for example:

Identity Object	(Class Code: 0x01)
Message Router Object	(Class Code: 0x02)
Assembly Object	(Class Code: 0x04)
Connection Object	(Class Code: 0x05)

Connection Manager Object	(Class Code: 0x06)
TCP/IP Interface Object	(Class Code: 0xF5)
EtherNet Link Object	(Class Code: 0xF6)

For the device-specific details, refer to the vendor documentation

Troubleshooting

The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.2.4 IL_ENIPSetAttributeSingle

Brief description

An individual attribute can be set or written using the **IL_ENIPSetAttributeSingle** attribute.

Target system	Library/folder
MLC	RIL_EtherNetIP/POUs/Acyclic_Communication
MTX	RIL_EtherNetIP/POUs/Acyclic_Communication

Tab. 7-12: Reference table

Interface description

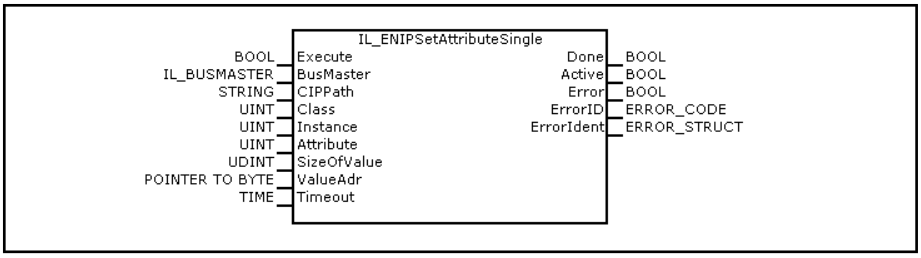


Fig. 7-4: Interface description of the IL_ENIPSetAttributeSingle function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string "aaa.bbb.ccc.ddd"
	Class	UINT	Class code
	Instance	UINT	Instance number
	Attributes	UINT	Attribute ID
	SizeOfValue	UDINT	Size of the data range addressed with ValueAdr
	ValueAdr	POINTER TO BYTE	Initial address of the data range
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error

Tab. 7-13: I/O interface of the IL_ENIPSetAttributeSingle function block

The input variables can assume the values specified in the following table.

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
CIPPath	STRING	Character string	Character string	„ “	Rising edge at "Execute"
Class	UINT	16#01	16#FF	16#01	Rising edge at "Execute"
Instance	UINT	16#00	16#FF	16#00	Rising edge at "Execute"
Attributes	UINT	16#01	16#FF	16#01	Rising edge at "Execute"
SizeOfValue	UDINT	16#0000	16#01C2	16#0000	Rising edge at "Execute"
ValueAdr	POINTER TO BYTE	N. def.	(450 bytes)	16#0000	Rising edge at "Execute"
Timeout	TIME	T#0ms	N. def.	T#2s	Rising edge at "Execute"

Tab. 7-14: Min./max. and default values of the IL_ENIPSetAttributeSingle inputs

Functional description

An individual attribute can be set or written using the **IL_ENIPSetAttributeSingle** attribute.

The device (adapter) is addressed via the IP address at first.

"Class", "Instance" and "Attribute" are used to access a certain "value" (or setting) of the adapter.

The attribute has a tree structure. However, class attributes are also available. In this case, the "Instance" is not specified or the "Instance" number is the attribute.

Addressing is manufacturer-specific. Usually, the following default objects are supported:

Identity Object	(Class Code: 0x01)
Message Router Object	(Class Code: 0x02)
Assembly Object	(Class Code: 0x04)
Connection Object	(Class Code: 0x05)
Connection Manager Object	(Class Code: 0x06)

TCP/IP Interface Object	(Class Code: 0xF5)
EtherNet Link Object	(Class Code: 0xF6)

For further information, refer to the manufacturer documentation and CIP specification

Troubleshooting The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.3 Backing up and restoring (Backup Restore)

7.3.1 IL_ENIPAdapterBackup

Brief description The function block **IL_ENIPAdapterBackup** is used to backup the configuration and the parameters of an Ethernet/IP adapter device to a file via acyclic reading (refer to chapter 7.2.1 "IL_ENIPGetAttributeSingle" on page 218).

Target system	Library/folder
MLC	RIL_ProfinetIO/POUs/Backup Restore

Tab. 7-15: Reference table

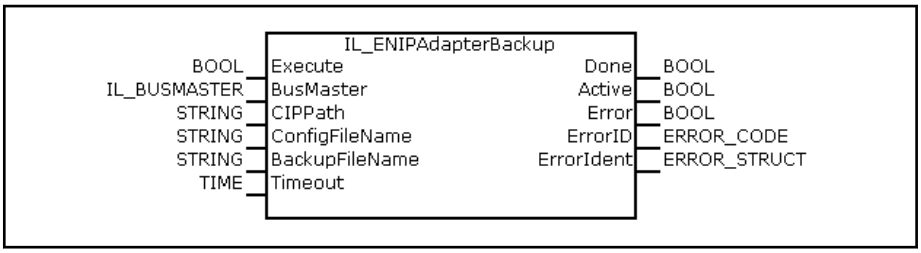


Fig. 7-5: Interface description of the IL_ENIPAdapterBackup function block

I/O type	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the scanner in the order of the configuration tree If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Also refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string "aaa.bbb.ccc.ddd"
	ConfigFileName	STRING	ConfigFile with "Class", "Instance" and "Attribute" of the parameters to be retrieved from the device. Name of configuration file without extension (*.enc)
	BackupFileName	STRING	Name of backup file without extension (*.enb)
	Timeout	TIME	Timeout monitoring of the function block

I/O type	Name	Type	Comment
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnostics in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE) Refer to "Error handling" on page 229

Tab. 7-16: I/O interface of the IL_ENIPAdapterBackup function block

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	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
Timeout	TIME	T#0ms		T#2s	Rising edge at "Execute"

Tab. 7-17: Min./max. and default values of the IL_ENIPAdapterBackup inputs

Functional description

The function block **IL_ENIPAdapterBackup** is used to backup the configuration and the parameters of an Ethernet/IP adapter device to a file after acyclic reading (refer to chapter 7.2.1 "IL_ENIPGetAttributeSingle" on page 218) from the adapter.

The function block retrieves the parameters listed in the configuration file "ConfigFileName" from the adapter and saves the data to the file "BackupFileName".

The configuration and backup files are stored on the Compact Flash card of the control (USER-Partition, /ata0b/).

The function block creates a new file upon each execution. If a file already exists, it is overwritten.

The configuration file is stored as text with the extension "*.enc".

Basic structure of the ConfigFile	Example of the ConfigFile
Class, Instance, Attribute	16#00C7, 16#0002, 16#0001
Class, Instance, Attribute	16#00C7, 16#0003, 16#0001
Class, Instance, Attribute	16#00C7, 16#0004, 16#0001
...	...
...	...
...	...
Class, Instance, Attribute	16#00C7, 16#0008, 16#0001

The function block retrieves the parameters from the adapter and saves them to the backup file at a rising edge at the "Execute" input.

Basic backup file	Example of the backup file
Class, Instance, Attribute, DataSize, Data	16#00C7, 16#0002, 16#0001, 16#0001, 16#58
Class, Instance, Attribute, DataSize, Data1, Data2	16#00C7, 16#0003, 16#0001, 16#0002, 16#20, 16#82
...	...
...	...
Class, Instance, Attribute, DataSize, Data1, Data2	16#00C7, 16#0004, 16#0001, 16#0002, 16#45, 16#56

Example program

This example describes the creation of the backup of the ENIP adapter parameters for storage on the Compact Flash card in the IndraControl control.

- Specify the name of the configuration file "strConfigFileName"
- Save the configuration file on the Compact Flash card (USER partition, /ata0b/) of the control

Steps to create and store the configuration file

1. Open notepad and enter backup parameter information.

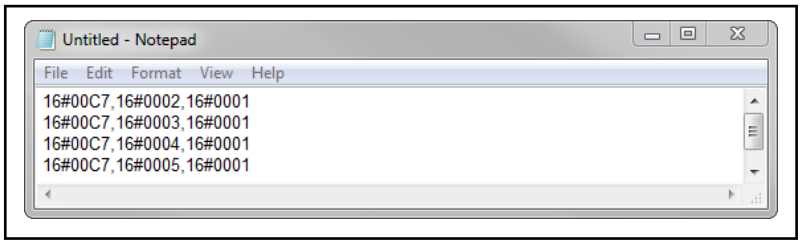


Fig. 7-6: Configuration file

2. Save configuration file as "ConfigFile.enc".
3. Right-click on the control in the Project Explorer and navigate to **Device data ▸ Explorer**.

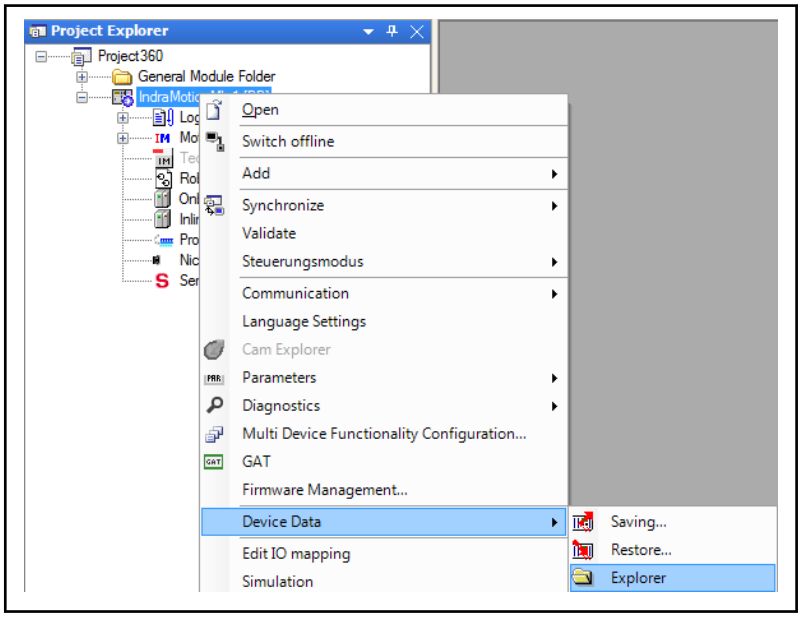


Fig. 7-7: Context menu of the control

4. Navigate to the user partition on the Compact Flash card of the control.

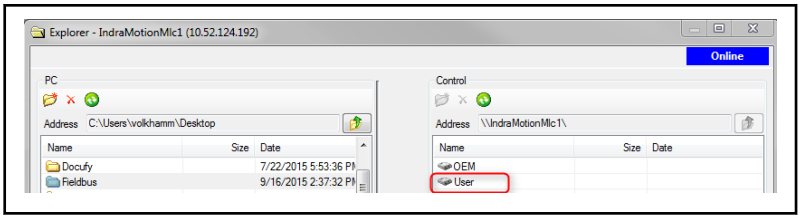


Fig. 7-8: User partition

5. Transfer of the configuration file to the user partition.

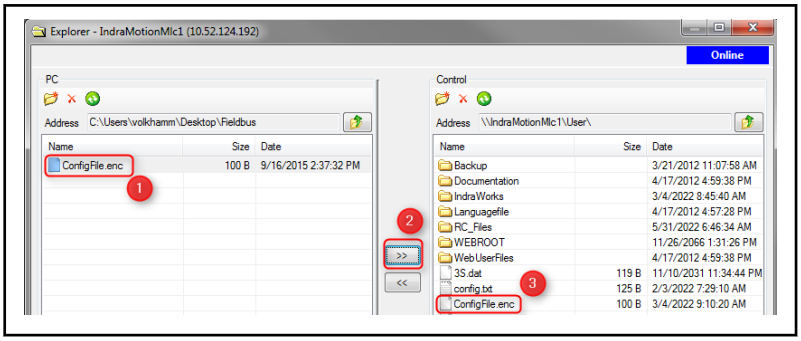


Fig. 7-9: Transfer in three steps

- Specify the name of the backup file "strBackupFileName"
- Enter the file name without extension, e.g. "Backupfile"
- Processing starts at a rising edge at the "bExecute" input:
  - Reading the parameters specified by the configuration file of the adapter addressed by "CIPPath"
  - Storing data in the backup file

Declaration

```
PROGRAM ProgIL_ENIPAdapterBackup
VAR
    fbIL_ENIPAdapterBackup: IL_ENIPAdapterBackup;
    bExecute: BOOL;                                 // FB running when TRUE
    strCIPPath: STRING := '192.168.2.2';            // IP addr. of ENIP adapter
    strConfigFileName: STRING:= 'ConfigFile';       // Config file name
    strBackupFileName: STRING:= 'BackupFile';       // Backup file name
    tTimeout: TIME := T#2S;                         // Timeout monitoring of FB
    bDone: BOOL;                                    // FB completed its task
    bActive: BOOL;                                  // FB working on actual task
    bError: BOOL;                                   // Indicates an error
    enErrorID: RIL_COMMONTYPES.ERROR_CODE;          // Class of error
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT; // Detailed info on error
END_VAR
```

Implementation

```
fbIL_ENIPAdapterBackup(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    CIPPath:= strCIPPath,
    ConfigFileName:= strConfigFileName,
    BackUpFileName:= strBackupFileName,
    Timeout:= tTimeout);
```

Error handling The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.3.2 IL_ENIPAdapterRestore

Brief description The function block **IL_ENIPAdapterRestore** is used to restore the parameters of an Ethernet/IP adapter device via acyclic writing from the backup file (chapter 7.2.4 "IL_ENIPSetAttributeSingle" on page 224).

Target system	Library/folder
MLC	RIL_EthernetIP/POUs/Backup Restore

Tab. 7-18: Reference table

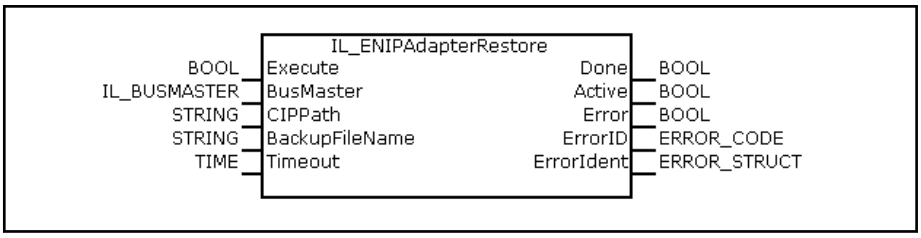


Fig. 7-10: Interface description of the IL_ENIPAdapterRestore function block

I/O type	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Also refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string "aaa.bbb.ccc.ddd"
	BackupFileName	STRING	Name of backup file without extension (*.enb)
	Timeout	TIME	Timeout monitoring of the function block
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing running, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Describes the diagnostics in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics (error table PROFINETIO_TABLE) "Error handling" on page 232

Tab. 7-19: I/O interface of the IL_ENIPAdapterRestore function block

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	Application
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
CIPPath	STRING	Character string	Character string	‘ ’	Rising edge at "Execute"

Name	Type	Min. value	Max. value	Default value	Application
BackupFileName	STRING	Character string	Character string	‘ ’	Rising edge at "Execute"
Timeout	TIME	T#0ms	--	T#2s	Rising edge at "Execute"

Tab. 7-20: Min./max. and default values of the IL_ENIPAdapterRestore inputs

Functional description

The function block **IL_ENIPAdapterRestore** is used to restore the parameters of an Ethernet/IP adapter device via acyclic writing from the backup file (chapter 7.2.4 "IL_ENIPSetAttributeSingle" on page 224).

The function block retrieves the parameters from the backup file of the Ethernet/IP adapter device stored on the Compact Flash card of the control.

The adapter is addressed via "CIPPath".

"Class", "Instance" and "Attribute" are used to address an individual "value" of the adapter.

The function block retrieves all parameters from the backup file at a rising edge at "Execute" and writes the parameter values to the adapter at the locations specified by "Slot", "SubSlot" and "Index".

The following table contains the basic structure and a backup file example.

Basic structure of the backup file	Example of the backup file
Class, Instance, Attribute, DataSize, Data	16#00C7, 16#0002, 16#0001, 16#0001, 16#58
Class, Instance, Attribute, DataSize, Data1, Data2	16#00C7, 16#0003, 16#0001, 16#0002, 16#20, 16#82
...	...
...	...
...	...
Class, Instance, Attribute, DataSize, Data1, Data2	16#00C7, 16#0004, 16#0001, 16#0002, 16#45, 16#56



The variable at the "BackupFileName" input should only consist of the file name without extension.

The extension (*.enb) is managed internally.

Example program

This example project describes the restoration of ENIP adapter device parameters saved to the backup file on the Compact Flash card in the IndraControl device.

The IP address of the ENIP adapter "CIPPath" and the file name of the backup file without extension have to be specified.

At a rising edge at the "bExecute" input, the process is executed: Parameter reading from the backup file and writing to the adapter addressed by the "CIPPath".

Declaration

```
PROGRAM ProgIL_ENIPAdapterRestore
VAR
    fbIL_ENIPAdapterRestore: IL_ENIPAdapterRestore;
    bExecute: BOOL                              // FB running when TRUE
    strCIPPath: STRING := '192.168.2.2';        // IP addr. of ENIP adapter
    strFileName: STRING:= 'BackupFile';         // Backup file name
    tTimeout: TIME := T#2S;                     // Timeout monitoring of FB
    bDone: BOOL;                                // FB completed its task
    bActive: BOOL;                              // FB working on actual task
    bError: BOOL;                               // Indicates an error
    enErrorID: RIL_COMMONTYPES.ERROR_CODE;      // Class of error
```

```
    enErrorIdent: RIL_COMMONTYPES.ERROR_STRUCT; // Detailed info on error
END_VAR
```

Implementation

```
fbIL_ENIPAdapterRestore(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> enErrorIdent,
    BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
    CIPPath:= strCIPPath,
    BackUpFileName:= strFileName,
    Timeout:= tTimeout);
```

Error handling The function block **IL_ENIPAdapterRestore** uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.4 Diagnostics (Diagnosis)

7.4.1 IL_ENIPRemoteAdapterState

Brief description The **IL_ENIPRemoteAdapterState** function block provides the simple diagnostics of an adapter.

Target system	Library/folder
MLC	RIL_EtherNetIP/POUs/Diagnosis
MTX	RIL_EtherNetIP/POUs/Diagnosis

Tab. 7-21: Reference table

Interface description

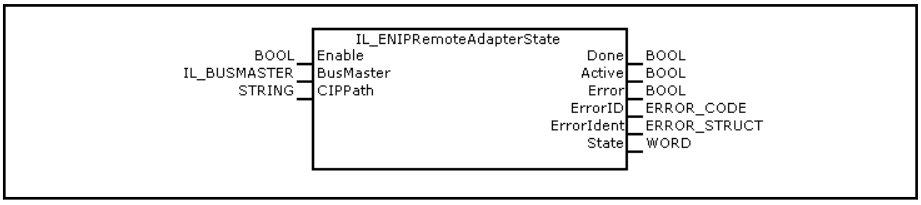


Fig. 7-11: Interface description of the IL_ENIPRemoteAdapterState function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Ethernet/IP scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string "aaa.bbb.ccc.ddd" E.g. IP addr.: "192.168.15.8"

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Function block processing enabled (once, edge-controlled)
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	State	WORD	State information of the adapter at the field bus (refer to "State output" on page 233)

Tab. 7-22: I/O interface of the IL_ENIPRemoteAdapterState function block

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Device does not exist or cannot be reached
1	IL_RBS_SLAVE_NOT_READY	Device not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Device reports configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Device reports available diagnostics
4	IL_FBS_SLAVE_ERROR	General error at device
5 …15	--- None ---	Reserved

Tab. 7-23: Bit assignment of the "State" output

The input variables can assume the values specified in the following table.

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Enable"
CIPPath	STRING	0.0.0.0	255.255.255.255	0.0.0.0	Rising edge at "Enable"

Tab. 7-24: Min./max. and default values of the IL_ENIPRemoteAdapterState inputs

Functional description

When activated, this function block determines the current status of the adapter.

This simple diagnostics queries this adapter error if there is a bus master diagnostics with a slave diagnostics or an adapter error.

The diagnostic information is output as data structure with defined bits.

Troubleshooting

The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.4.2 IL_ENIPRemoteAdapterStateDetails

Brief description

The **IL_ENIPRemoteAdapterStateDetails** function block provides the detailed diagnostics of an adapter. Attribute 5 "Status" is retrieved from the "Identity object" by the adapter.

Target system	Library/folder
MLC for 14V06 and higher	RIL_EtherNetIP/POUs/Diagnosis
MTX from 14V06	RIL_EtherNetIP/POUs/Diagnosis

Tab. 7-25: Reference table

Interface description

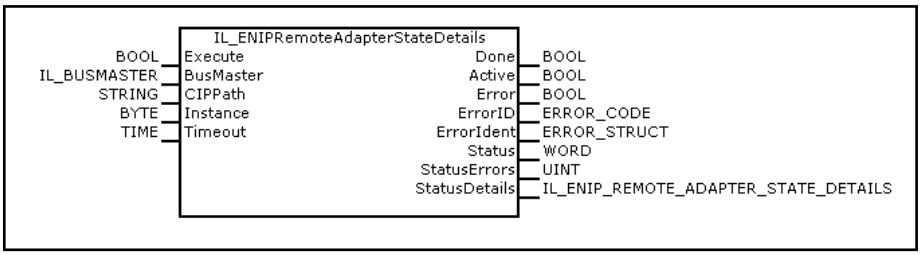


Fig. 7-12: Interface description of the IL_ENIPRemoteAdapterStateDetails function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Ethernet/IP scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
	CIPPath	STRING	Address of CIP device as string. Structure: "aaa.bbb.ccc.ddd" E.g. IP addr.: "192.168.15.8"
	Instance	BYTE	Instance number; generally, one adapter has only one "Identity Object" (instance = 1)
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid.
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	Status	WORD	Detailed status of the Ethernet/IP adapter (corresponds to attribute 5 of the "Identity Object") Refer to tab. 7-28 "Description of the Status attribute of the Identity Object" on page 235
	StatusErrors	UINT	Error bits (bits 8 -11) from "Status"
	StatusDetails	IL_ENIP_REMOTE_ADAPTER_STATE_DETAILS	Detailed status (bit 4 - 7) from "Status" Refer to chapter 7.5.3 "IL_ENIP_REMOTE_ADAPTER_STATE_DETAILS" on page 240

Tab. 7-26: I/O interface of the IL_ENIPRemoteAdapterStateDetails function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"
CIPPath	STRING	„0.0.0.0“	"255.255.255.255"	"0.0.0.0"	Rising edge at "Execute"
Instance	BYTE	0	255	1	Rising edge at "Execute"
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at "Execute"

Tab. 7-27: Min./max. and default values of the IL_ENIPRemoteAdapterStateDetails inputs

Functional description

When activated, this function block **IL_ENIPRemoteAdapterStateDetails** determines the extended status of the adapter once. Attribute 5 "Status" is retrieved from the "Identity object" by the adapter:

Bit	Naming	Description
0	Owned	TRUE: The adapter (or an object in the adapter) is assigned by a scanner
1	-	Reserved
2	Configured	TRUE: The application in the adapter is configured (i.e. not upon delivery anymore)
3	-	Reserved
4-7	Extended Device Status	See below. A vendor-specific definition is also possible
8	Minor Recoverable Fault	TRUE: There is a minor, recoverable error in the device. The device is still ready for operation
9	Minor Unrecoverable Fault	TRUE: There is a minor, unrecoverable error in the device. The device is still ready for operation
10	Major Recoverable Fault	TRUE: There is a fatal, recoverable error in the device. The device is error state
11	Major Unrecoverable Fault	TRUE: There is a fatal, unrecoverable error in the device. The device is error state
12-15	Extended Device Status 2	Reserved or vendor-specific implementation

Tab. 7-28: Description of the "Status" attribute of the "Identity Object"

Value	Description
0	Self-test or unknown
1	Firmware update active
2	At least one I/O connection in error state
3	No I/O connection established
4	Saved configuration corrupt
5	Fatal error: Bit 10 or 11 in "Status" is TRUE
6	At least one I/O connection is active

Value	Description
7	At least one I/O connection is established. All are in the state "Idle" (invalid data)
8-9	Reserved
10-15	Vendor-specific

Tab. 7-29: "Extended Device Status" (bits 4-7) in the "Status" attribute

Troubleshooting

The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.4.3 IL_ENIPScannerState

Brief description

The **IL_ENIPScannerState** function block provides the simple diagnostics of the Ethernet/IP scanner.

This simple diagnostics can be cyclically queried in a polling. Only when the function block reports a diagnostics, the more complex function block **IL_ENIPRemoteAdapterStateDetails** (refer to chapter 7.4.4 "IL_ENIPScannerStateDetails" on page 237) should be used to get a detailed diagnostics.

Target system	Library/folder
MLC	RIL_EtherNetIP/POUs/Diagnosis
MTX	RIL_EtherNetIP/POUs/Diagnosis

Tab. 7-30: Reference table

Interface description

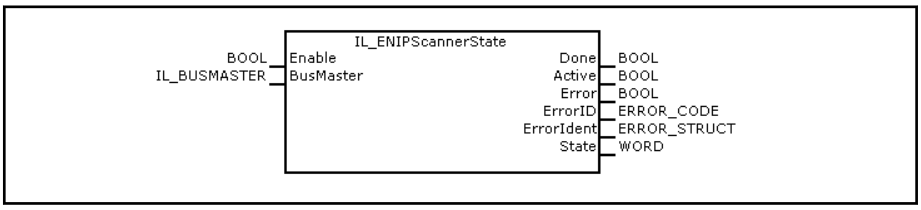


Fig. 7-13: Interface description of the IL_ENIPScannerState function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function block enabled (permanent, state-controlled)
	BusMaster	IL_BUSMASTER	Scanner ID. If several scanners are operated simultaneously, they are differentiated with the "BusMaster" variable Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	ERROR_CODE from RIL_CommonTypes. Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Error structure from the ETHERNET_IP_TABLE error table
	State	WORD	State of the Ethernet/IP scanner (refer to "State output" on page 236)

Tab. 7-31: I/O interface of the IL_ENIPScannerState function block

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_FBM_BUS_OFF	Field bus not ready
1	IL_FBM_MASTER_FAILURE	Error of the bus master
2	IL_FBM_SLAVE NOT_REACHABLE	One or several field bus devices cannot be reached
3	IL_FBM_SLAVE_DIAGNOSIS	One or several field bus devices report a diagnostics
4	IL_FBM_SLAVE_ERROR	One or several field bus devices report an error
5 …15	--- None ---	Reserved

Tab. 7-32: Bit assignment of the "State" output

The input variables can assume the values specified in the following table.

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Enable"

Tab. 7-33: Min./max. and default values of the IL_ENIPScannerState inputs

Functional description

A simple diagnostics of the scanner can be read with the **IL_ENIPScanner-State** function block.

The simple diagnostics can be queried cyclically. Only when the function block reports a diagnostics, the more complex **IL_ENIPRemoteAdapterState-Details** function block should be used to get a detailed diagnostics.

The diagnostic information is output as bit array with defined bits.

The "State" output indicates the master state.

If there is no error in communication, the value 16#0000 is returned at the "State" output.

If the "State" output is not 16#0000, an error reaction can be initiated according to the output bits defined in the user program.

Troubleshooting

The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.4.4 IL_ENIPScannerStateDetails

Brief description

The following detailed diagnostics of the scanner can be read using the **IL_ENIPScannerStateDetails** function block:

- State of the communication stack
- State of the bus communication
- Error counter
- Overview on configured, active adapters reporting a diagnostics

Target system	Library/folder
MLC	RIL_EtherNetIP/POUs/Diagnosis
MTX	RIL_EtherNetIP/POUs/Diagnosis

Tab. 7-34: Reference table

Interface description

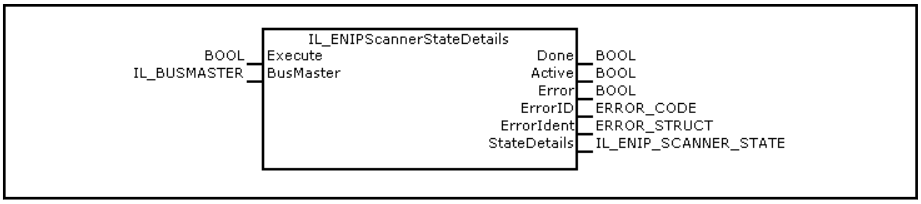


Fig. 7-14: Interface description of the IL_ENIPScannerStateDetails function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the Ethernet/IP scanner in the order of the configuration tree. If only one EtherNet/IP scanner is available, the value of this scanner is always IL_BUSMASTER_0 Refer to chapter 7.5.1 "IL_BUSMASTER" on page 239
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	StateDetails	IL_ENIP_SCANNER_STATE	Data structure of the scanner status Refer to chapter 7.5.4 "IL_ENIP_SCANNER_STATE" on page 241

Tab. 7-35: I/O interface of the IL_ENIPScannerStateDetails function block

The input variables can assume the values specified in the following table.

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute"

Tab. 7-36: Min./max. and default values of the IL_ENIPScannerStateDetails inputs

Functional description

The following detailed diagnostics of the scanner can be read using the **IL_ENIPScannerStateDetails** function block:

- State of the communication stack
- State of the bus communication
- Error counter and
- Overview on configured, active adapters reporting a diagnostics

This function block should be called if an error has been reported during the simple diagnostics.

Troubleshooting

The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151), refer to chapter 7.5.7 "Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)" on page 242.

7.5 Data types (DataTypes)

7.5.1 IL_BUSMASTER

Brief description

The enumeration type **IL_BUSMASTER** of the RIL_FieldbusTypes.library is used to select the respective Ethernet/IP scanner. In addition to the onboard real-time Ethernet interface, other scanners can also be available in an MLC system via function modules.

The Ethernet/IP scanners are distinguished based on their ascending order in the configuration.

Indexes are assigned to the controllers in the following order:

- Onboard master
- Function module 1
- Function module 2
- Function module 3
- Function module 4

☞ If, for example, the onboard master is no Ethernet/IP scanner while an Ethernet/IP scanner is configured on function module 1,the value IL_BUSMASTER_0 is assigned to the controller on function module.

Element	Value	Description
IL_BUSMASTER_0	0	First Ethernet/IP scanner
IL_BUSMASTER_1	1	Second Ethernet/IP scanner
IL_BUSMASTER_2	2	Third Ethernet/IP scanner
IL_BUSMASTER_3	3	Fourth Ethernet/IP scanner
IL_BUSMASTER_4	4	Fifth Ethernet/IP scanner

Tab. 7-37: IL_BUSMASTER enumeration type

7.5.2 IL_ENIP_IDENTITY_ATTRIBUTES

The **IL_ENIP_IDENTITY_ATTRIBUTES** structure contains the mandatory attributes of the "Identity Object" ready via "Get-Attribute-All".

Vendor ID	UINT	Vendor identification, e.g. Bosch Rexroth = 287
Device Type	UINT	Device type identification (from CIP specification)
Product Code	UINT	Identification of specified product (vendor-specific)
Major Revision	BYTE	Major Revision
Minor Revision	BYTE	Minor Revision
Status	WORD	Status (refer to the following tables)
Serial Number	UDINT	Serial number
Product Name	STRING(32)	Product name

Tab. 7-38: Mandatory attribute of the "Identity Object"

Bit	Naming	Description
0	Owned	TRUE: The adapter (or an object in the adapter) is assigned by a scanner
1	-	Reserved
2	Configured	TRUE: The application in the adapter is configured (i.e. not upon delivery anymore)
3	-	Reserved
4-7	Extended Device Status	See below. A vendor-specific definition is also possible
8	Minor Recoverable Fault	TRUE: There is a minor, recoverable error in the device. The device is still ready for operation
9	Minor Unrecoverable Fault	TRUE: There is a minor, unrecoverable error in the device. The device is still ready for operation
10	Major Recoverable Fault	TRUE: There is a fatal, recoverable error in the device. The device is error state
11	Major Unrecoverable Fault	TRUE: There is a fatal, unrecoverable error in the device. The device is error state
12-15	Extended Device Status 2	Reserved or vendor-specific implementation

Tab. 7-39: Description of the "Status" attribute of the "Identity Object"

Value	Description
0	Self-test or unknown
1	Firmware update active
2	At least one I/O connection in error state
3	No I/O connection established
4	Saved configuration corrupt
5	Fatal error: Bit 10 or 11 in "Status" is TRUE
6	At least one I/O connection is active
7	At least one I/O connection is established. All are in the state "Idle" (invalid data)
8-9	Reserved
10-15	Vendor-specific

Tab. 7-40: "Extended Device Status" (bits 4-7) in the "Status" attribute

7.5.3 IL_ENIP_REMOTE_ADAPTER_STATE_DETAILS

The enumeration type **IL_ENIP_REMOTE_ADAPTER_STATE_DETAILS** allows to display the bits 4...7 "Detailed status" of an Ethernet/IP adapter (corresponds to attribute 5 of the "Identity Object") as text.

Name	Value	Description
SelfTestingOrUnknown	0	Self-test or unknown
FwUpdateInProgress	1	Firmware update running
AtLeastOneFaultedIoConnection	2	At least one incorrect I/O connection
NoIoConnectionEstablished	3	No I/O connection established

Name	Value	Description
NonVolatileConfigurationBad	4	
MajorFault	5	
AtLeastOneIoConnectionInRun	6	At least one I/O connection in the "Run" state
AtLeastOneIoConnectionInIdle	7	At least one I/O connection in the "Idle" state
Reserved_8	8	Reserved
Reserved_9	9	Reserved
Vendor_Specific_10	10	Vendor-specific_10
Vendor_Specific_11	11	Vendor-specific_11
Vendor_Specific_12	12	Vendor-specific_12
Vendor_Specific_13	13	Vendor-specific_13
Vendor_Specific_14	14	Vendor-specific_14
Vendor_Specific_15	15	Vendor-specific_15

Tab. 7-41: Detailed status (bit 4 - 7) from "Status".

7.5.4 IL_ENIP_SCANNER_STATE

Brief description The data structure **IL_ENIP_SCANNER_STATE** contains the diagnostic information on:

- State of the communication stack
- State of the bus communication
- Error counter
- Overview on configured, active slaves reporting a diagnostics

Structural element	Type	Description
CommunicationCOS	DWORD	Status of the communication stack, see below
CommunicationState	IL_FBUS_COMMUNICATION_STATE	Status of the communication See chapter 5.8.2 "IL_FBUS_COMMUNICATION_STATE" on page 185
Version	UINT	Version of the data structure of the diagnostic information
Watchdog	UINT	Watchdog timeout between PLC and field bus connection in ms
ErrorCount	UDINT	Error counter. Counts the number communication errors that have occurred (both internal errors and communication errors)
SlaveState	IL_FBUS_SLAVE_STATE	Indicates whether the scanner has an active connection to all configured slaves Refer to chapter 5.8.3 "IL_FBUS_SLAVE_STATE" on page 185
NumOfConfigSlaves	UDINT	Number of configured slaves
NumOfActiveSlaves	UDINT	Number of active slaves, i.e., slaves to which the master has established a communication connection
NumOfDiagSlaves	UDINT	Number of slaves with an active diagnostics

Tab. 7-42: IL_ENIP_SCANNER_STATE data structure

The state of the communication stack is binary-encoded in the "CommunicationCOS" output.

The information contained in the stack is usually only required if problems occur and is interpreted by Bosch Rexroth. The bits are contained in the library in the form of defines (ENIP_Defines).

Output "CommunicationCOS"

Bit no.	Name	Description
0	COMM_COS_READY	Protocol stack has been started and waits for configuration
1	COMM_COS_RUN	Protocol stack has been configured and waits for bus connection
2	COMM_COS_BUS_ON	Protocol stack may communicate on the bus
3	COMM_COS_CONFIG_LOCKED	Bus configuration is locked and cannot be changed
4	COMM_COS_CONFIG_NEW	New bus configuration is available but not initialized yet
5	COMM_COS_RESTART_REQUIRED	Communication restart required
6	COMM_COS_RESTART_REQUIRED_ENABLE	Communication restart possible
....		Remaining values: FALSE

7.5.5 IL_FBUS_COMMUNICATION_STATE

Brief description The enumeration type **IL_FBUS_COMMUNICATION_STATE** is used to display the status of the field bus communication.

Element	Value	Description
COMM_STATE_NOT_CONFIGURED	16#00000001	Communication is not configured
COMM_STATE_STOP	16#00000002	Communication is in "Stop" state
COMM_STATE_IDLE	16#00000003	Communication is in "Idle" state
COMM_STATE_OPERATE	16#00000004	Communication in cyclic operation

Tab. 7-43: IL_FBUS_COMMUNICATION_STATE enumeration type

7.5.6 IL_FBUS_SLAVE_STATE

The enumeration type **IL_FBUS_SLAVE_STATE** is used to show the status of the communication relations to the slaves. That means, that the status indicates whether the field bus master has an active connection to all configured slaves.

Element	Value	Description
SLAVE_STATE_OK	16#00000001	Communication to all active slaves available
SLAVE_STATE_FAILED	16#00000002	Communication error to at least one slave

Tab. 7-44: IL_FBUS_SLAVE_STATE enumeration slave

7.5.7 Excerpt from the error table ETHERNET_IP_TABLE (ERROR_TABLE = 16#0151)

The following error codes apply to the function blocks of the RIL_EtherNetIP library.

The error codes are reported as ERROR_STRUCT with the elements "Additional1" and "Additional2" at the "ErrorIdent" output.

ErrorID	Additional1	Additional2	
INPUT_INVALID_ERROR	16#00000001	16#00000000	"BusMaster" input is invalid
		16#00000001	"CIPPath" input is invalid
		16#00000002	"ServiceCode" input may not be "0"
		16#00000003	"Class" input may not be "0"
		16#00000004	"Attribute" input may not be "0"
		16#00000005	"SizeOfValue" and "ValueAdr" inputs may not be "0"
		16#00000006	The data range at the inputs "ValueAdr" / "SizeOfValue" is too small for the received data
		16#00000007	There is an invalid input parameter (internal error)
SYSTEM_ERROR	16#00000002	(Internal)	Internals error (interface pointer)
COMMUNICATION_ERROR	16#00000003	(Internal)	Check the input parameters "ServiceCode", "Class" and "Attribute". The addressed CIP object might not be supported by the adapter. Internal error (netX), also refer to tab. 7-46 "Special error codes of the EtherNet/IP_Table (ERROR_TABLE = 16#0151)" on page 244
SYSTEM_ERROR	16#00000101	(Internal)	Internal error (scanner not found)
SYSTEM_ERROR	16#00000102	(Internal)	Internal error (scanner status)
SYSTEM_ERROR	16#00000103	(Internal)	Internal error (I/O driver not ready)
COMMUNICATION_ERROR	16#00000104	(Internal)	Timeout: a.) check the address at the "CIPPath" input (or the addressed device cannot be reached) b.) increase the value at the "Timeout" input
STATE_MACHINE_ERROR	16#00000105	(Internal)	Internal error (state machine)
SYSTEM_ERROR	16#00000106	(Internal)	Internals error (interface not enabled)
COMMUNICATION_ERROR	16#00000107	(Internal)	Internal error (retry)
INPUT_RANGE_ERROR	16#00000108	(Internal)	The adapter was not found. Check the "CIPPath" input
SYSTEM_ERROR	16#00000109	(Internal)	Internal error (parameter not found)
CALCULATION_ERROR	16#00000110	(Internal)	Internal error (computation error)
RESOURCE_ERROR	16#00000111	(Internal)	Reserved buffer too small
INPUT_INVALID_ERROR	16#3002	16#0002	Invalid name of the backup file
ACCESS_ERROR	16#3003	16#0002	Invalid format of the backup file
ACCESS_ERROR	16#3003	16#0003	Backup file cannot be opened
ACCESS_ERROR	16#3003	16#0004	Reading or writing the backup file is not possible

Tab. 7-45: Error codes of the RIL_EtherNetIP

The following table contains special "Additional2" error codes that can occur in case of the internal error "Additional1" = 16#0003.

Additional2	
16#C01F0034	Ethernet/IP object task upon reset
16#C00800C8	Task is not initialized
16#C00800C9	Task is busy (internal)
16#C00800D2	Configuration data basis not available
Error codes tool kit	
16#800A0012	Reception data buffer too small
16#800C0017	Message buffer for sending messages is full
16#800C0019	No netX message received
16#800C001A	Message buffer too small for the incoming message

Tab. 7-46: Special error codes of the EtherNet/IP_Table (ERROR_TABLE = 16#0151)

8 RIL_EtherNetIPAdapter.library

8.1 Introduction and overview

The controls can be operated as field bus slaves. This special field bus slave is called EtherNet/IP adapter in the context of EtherNet/IP.

The present library describes the interfaces of the EtherNet/IP adapter to the IEC user program. EtherNet/IP devices can be diagnosed via these interfaces.

The IL_EnableDoneBase and IL_ExecuteDoneBase methods are used and extended or overwritten

Two interface types are supported:

- EtherNet/IP adapter via the engineering interface
- EtherNet/IP adapter via the Onboard or function module interface

Areas of application and function blocks of the RIL_EtherNetIPAdapter.library

Area of application (folder in library)	
Function blocks (POUs)	**Description**
Diagnostics (Diagnosis), via the onboard or function module interface	
IL_ENIPAdapterState, page 245	Diagnostics: Basic adapter state
IL_ENIPAdapterStateDetails, page 247	Diagnostics: Detailed status of the Ethernet/IP stack of the adapter
Diagnostics (Diagnosis/EngineeringPortAdapter), via the engineering interface	
IL_Status, page 248	Diagnostics of the cyclic communication.
With data types:	
IL_ENIP_ADAPTER_STATE, page 250	Structure, containing diagnostic information on the state of the communication stack, the state of the bus communication, the error counter as well as an overview on configured, active and slaves reporting diagnostics.

Tab. 8-1: RIL_EtherNetIPAdapter library

Selecting the slave

The IL_BUSSLAVE enumeration type of the RIL_FieldbusTypes library selects the appropriate EtherNet/IP adapter.

Also refer to IL_BUSSLAVE, page 250

8.2 Diagnostics (Diagnosis)

8.2.1 IL_ENIPAdapterState

Brief description

This function block returns a basic diagnosis of the EtherNet/IP adapter.

This simple diagnostics can be cyclically queried in a polling.

Target system	Library/folder
MLC	RIL_EtherNetIPAdapter/ POUs/ Diagnosis
MTX	RIL_EtherNetIPAdapter/ POUs/ Diagnosis

Tab. 8-2: Reference table

Interface description

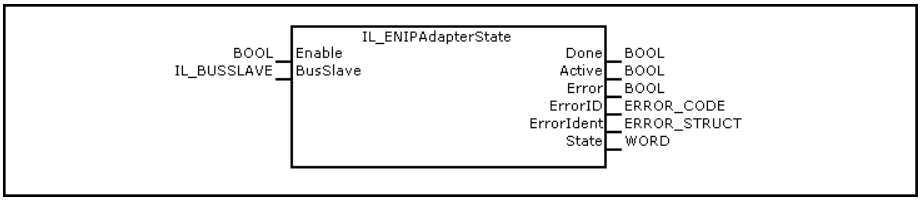


Fig. 8-1: Interface of FB IL_ENIPAdapterState

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the adapter in the order of the configuration tree. If only one EtherNet/IP adapter is available, the value of this adapter is always IL_BUSSLAVE_0. See IL_BUSSLAVE on page 250.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable ETHERNETIP_TABLE)
	State	WORD	State of the EtherNet/IP adapter, see below

Tab. 8-3: I/O interface of FB IL_ENIPAdapterState

The "State" output provides a rough diagnostic classification in binary code.

The bits are contained in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Output "State"

Bit no.	Name	Description
0	IL_FBS_BUS_OFF	No field bus communication
2	IL_FBS_CONFIG_ERROR	Faulty configuration by the field bus master
4	IL_FBS_SLAVE_ERROR	Device error

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Enable

Tab. 8-4: Min./max. and default values of the IL_ENIPAdapterState inputs

Functional description

The function block returns the state information of the EtherNet/IP adapter as bit array in "State".

The simple diagnostics is queried cyclically. Only when the function block reports a diagnostics, the IL_ENIPAdapterStateDetails, page 247, function block should be used to get a detailed diagnostics.

Troubleshooting

The function block uses the **error table ETHERNET_IP_TABLE** (ERROR_TABLE = 16#0151). It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	Internal use	Adapter not found in the configuration table

Tab. 8-5: FB IL_ENIPAdapterState error codes

8.2.2 IL_ENIPAdapterStateDetails

Brief description

This function block determines the detailed status of the EtherNet/IP adapter stack. It provides information on

- state of the communication stack
- the state of the bus communication
- the previous error values

Target system	Library/folder
MLC	RIL_EtherNetIPAdapter/ POUs/ Diagnosis
MTX	RIL_EtherNetIPAdapter/ POUs/ Diagnosis

Tab. 8-6: Reference table

Interface description

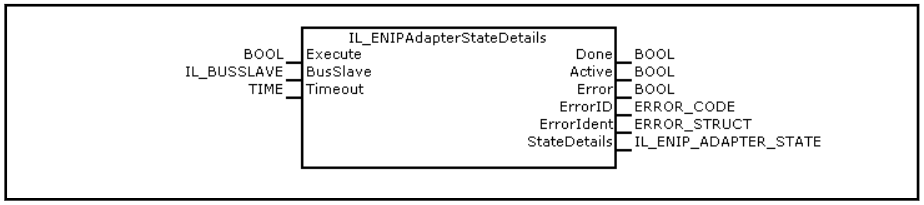


Fig. 8-2: Interface of FB IL_ENIPAdapterStateDetails

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the field bus slave in the order of the configuration tree. See IL_BUSSLAVE on page 250.
	Timeout	TIME	Timeout - Monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	AdapterState	IL_ENIP_ADAPTER_STATE	Detailed information about the status of the adapter stack Also refer to IL_ENIP_ADAPTER_STATE on page 250

Tab. 8-7: I/O interface of FB IL_ENIPAdapterStateDetails

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Execute
Timeout	TIME	T#0ms	T#3600s	T#2s	Rising edge at “Execute”

Tab. 8-8: Min./max. and default values of the IL_ENIPAdapterStateDetails inputs

Functional description

When activated, this function block determines the current state of the adapter once.

Troubleshooting

The function block uses error table **ETHERNETIP_TABLE** (ERROR_TABLE = 16#0151).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	Internal use	Scanner not in configuration table
COMMUNICATION_ERROR	16#0104	0	Timeout
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
COMMUNICATION_ERROR	16#0003	*EtherNet/IP adapter diagnostics* • General error codes, EtherNet/IP adapter , page 251, or • Special error codes, EtherNet/IP adapter , page 253..	

Tab. 8-9: FB IL_ENIPAdapterStateDetails error codes

8.3 Diagnostics (Diagnosis/EngineeringPortAdapter)

8.3.1 IL_Status

Brief description

This function block returns a basic diagnosis of the EtherNet/IP adapter (Engineering port).

This basic diagnostics can be cyclically queried in a polling.

Target system	Library/folder
MLC	RIL_EtherNetIPAdapter/ POUs/ Diagnosis/EngineeringPortAdapter
MTX	RIL_EtherNetIPAdapter/ POUs/ Diagnosis/EngineeringPortAdapter

Tab. 8-10: Reference table

Interface description

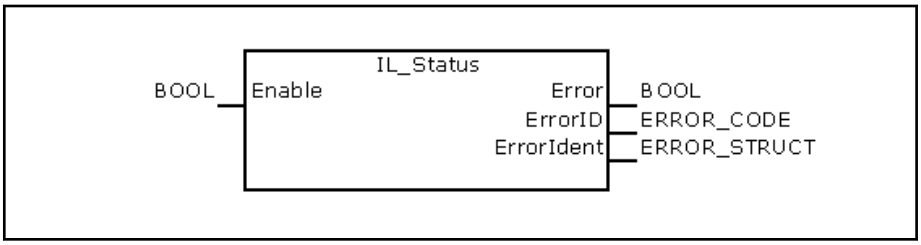


Fig. 8-3: IL_Status

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
VAR_OUTPUT	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable ETHERNETIP_TABLE)

Tab. 8-11: Interface of IL_Status

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous

Tab. 8-12: Min./max. and default values of the IL_Status inputs

Functional description

When Enable is TRUE, the state of the cyclic EtherNet/IP communication is queried whenever the function block is called.

If an error is detected in cyclic communication, the state of output "Error" becomes TRUE. The error message in Additional1 and Additional2 indicates the cause of the communication error.

Troubleshooting

The function block uses error table **ETHERNETIP_TABLE** (ERROR_TABLE = 16#0151).

It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
COMMUNICATION_ERROR	16#10000001	16#00000000	NoCyclicCom: The scanner has not yet established cyclic communication with the IndraLogic EtherNet/IP adapter.
COMMUNICATION_ERROR	16#10000002	16#00000000	Idle: The EtherNet/IP scanner has started cyclic communication but has set the Idle flag in the cyclic channel. In this way, the scanner signals that its output image is invalid.
COMMUNICATION_ERROR	16#10000003	16#00000000	Timeout: The cyclic communication time monitoring function signals an error.
COMMUNICATION_ERROR	16#10000004	16#00000000	Closed: Cyclic communication was actively completed by the EtherNet/IP scanner.

Tab. 8-13: Error codes of the IL_Status function block

Example

The following example shows the function block IL_Status is to be used.

Example

```
(*********************************************
** Variables for IL_Status
*********************************************)
PROGRAM PLC_PRG
VAR
  fbDiag:         IL_Status;
```

```
 diCtrValidInput: DINT;
  ...
END_VAR;
  ...
(***********************************************
** Program
***********************************************)
fbDiag(Enable:=TRUE);
IF(Diag.Error = TRUE) THEN
  (*Insert error handling here.*)
...
  fbDiag(Enable:=FALSE ); (*Error (Enable) reset*)
ELSE
  (*Valid data: Insert input/output data processing here *)
  diCtrValidInput := diCtrValidInput +1;
END_IF
...
```

8.4 Data types (DataTypes)

8.4.1 IL_BUSSLAVE

Brief description

The enumeration type IL_BUSSLAVE of the RIL_FieldbusTypes.library is used to select the respective field bus slave. In addition to the onboard real-time Ethernet interface, other EtherNet/IP adapters can also be available in a control via function modules.

The devices are distinguished based on their ascending order in the configuration. Indexes are assigned to the devices in the following order

- Onboard
- Function module 1
- Function module 2
- Function module 3
- Function module 4

Element	Value	Description
IL_BUSSLAVE_0	0	First device
IL_BUSSLAVE_1	1	Second device
IL_BUSSLAVE_2	2	Third device
IL_BUSSLAVE_3	3	Fourth device
IL_BUSSLAVE_4	4	Fifth device

Tab. 8-14: Elements of the IL_BUSSLAVE enumeration type

8.4.2 IL_ENIP_ADAPTER_STATE

Brief description

The IL_ENIP_ADAPTER_STATE data structure contains the status information of the EtherNet/IP adapter stack.

Structural element	Type	Description
LastGRC	UDINT	Latest CIP General Error Code, page 253
LastERC	UDINT	Latest Extended Error Code, page 255
CurrentConnection	UDINT	Established connections

Structural element	Type	Description
CommunicationState	UDINT	Network state of communication channel, see below
CommunicationError	UDINT	Current error code of the communication channel. The meaning of the error can be learned from the error table, see below.

Tab. 8-15: IL_ENIP_ADAPTER_STATE data structure

Element "CommunicationState" of data structure IL_ENIP_ADAPTER_STATE describes the state of communication of the EtherNet/IP adapter.

Structure element "CommunicationState"

Value	Name	Description
16#00000000	UNKNOWN	Communication state unknown
16#00000001	NOT_CONFIGURED	Communication channel was not configured
16#00000002	STOP	Stop
16#00000003	IDLE	Idle
16#00000004	OPERATE	EtherNet/IP communication

Element "CommunicationError" of data structure IL_ENIP_ADAPTER_STATE describes the current error code of the communication channel.

Structure element "CommunicationError"

Value	Name	Description
16#00000000	SUCCESS	Communication channel running
16#C0000140	NETWORK FAULT	Network fault
16#C0000141	CONNECTION CLOSED	Communication was closed
16#C0000142	CONNECTION TIMED OUT	The communication time monitoring function signals an error.
16#C0000143	LONELY NETWORK	Lonely network
16#C0000144	DUPLICATE NODE	Duplicate nodes
16#C0000145	CABLE DISCONNECT	Cable was pulled off

8.4.3 General error codes, EtherNet/IP adapter

The error codes presented are applicable to ENIP adapters. These error codes are general error codes, i.e., they are independent of any bus.

The error codes are signaled as ERROR_STRUCT, Additional2 with ERROR_STRUCT, **Additional1= 16#0003**.

Additional2	
16#800A0001	Invalid pointer (NULL) applied to driver
16#800A0002	No card available with the specified name/index
16#800A0003	No channel available with the specified index
16#800A0004	Invalid handle applied to driver
16#800A0005	Invalid parameters
16#800A0006	Invalid command

Additional2	
16#800A0007	Invalid buffer size
16#800A0008	Invalid access size
16#800A0009	Function failed
16#800A000A	File Could Not be Opened
16#800A000B	File size is zero
16#800A000C	Insufficient memory capacity to load the file
16#800A000D	Comparing file checksums failed
16#800A000E	Error when reading the file
16#800A000F	Invalid file type
16#800A0010	Invalid file name
16#800A0011	Driver function not available
16#800A0012	Specified buffer is too small
16#800A0013	Memory cannot be mapped
16#800A0014	No further entries present
16#800B0001	Driver not initialized
16#800B0002	Init status error of the driver
16#800B0003	Status read error of the driver
16#800B0004	Command is active on device
16#800B0005	General error upon download
16#800B0006	Wrong driver version
16#800B0030	CIFx driver not in operation
16#800B0031	Device could not be initialized
16#800B0032	Channel could not be initialized (xOpenChannel was not called)
16#800B0033	IOControl call failed
16#800B0034	Driver could not be opened
16#800C0010	The dual port memory could not be accessed (memory card not found)
16#800C0011	Device not ready (Ready flag failed)
16#800C0012	Device not ready for operation (Running flag failed)
16#800C0013	Watchdog test error
16#800C0015	Handshake flag error
16#800C0016	Mailbox for sent mails full
16#800C0017	Timeout for sending data packages
16#800C0018	Timeout for receiving data packages
16#800C0019	No data package available
16#800C001A	Insufficient memory space in mailbox
16#800C0020	Reset command timeout

Additional2	
16#800C0021	COM flag not set
16#800C0022	I/O data exchange failed
16#800C0023	I/O data exchange timeout
16#800C0024	Unknown I/O data exchange mode
16#800C0025	Device function failed
16#800C0026	DPM size “size” differs from the configuration
16#800C0027	Unknown status mode
16#800C0028	Output port is already used
16#800C0029	Timeout configuration lock
16#800C002A	Configuration unlock timeout
16#800C002B	Set HOST status timeout
16#800C002C	Delete HOST status timeout
16#800C002D	Timeout during channel initialization
16#800C002E	Set bus ON timeout
16#800C002F	Set bus OFF timeout

Tab. 8-16: General error codes

8.4.4 Special error codes, EtherNet/IP adapter

This section presents the special error codes of the EtherNet/IP adapter.

The error codes are signaled as ERROR_STRUCT, Additional2 with ERROR_STRUCT, **Additional1= 16#0003**.

Additional2	
16#C0590001	Invalid command
16#C0590004	Incorrect TCP/IP configuration
16#C0590009	Invalid offset for I/O data

Tab. 8-17: EtherNet/IP adapter, special error codes

8.4.5 CIP, general error codes

This section introduces the CIP general error codes.

Code	
16#01	A connection-related service failed along the connection path
16#02	Resources needed for the object to perform the requested service were unavailable
16#03	See status code 0x20 which is the preferred value to use for this condition.
16#04	A path segment error has been encountered. Evaluation of the supplied path information failed.
16#05	The path references an unknown object class, instance or structure element causing the abort of path processing.
16#06	Only a part of the expected data could be transferred.
16#07	The messaging connection was lost.

Code	
16#08	The requested service has not been implemented or has not been defined for this object class or instance.
16#09	Detection of invalid attribute data
16#0A	An attribute in the Get_Attribute_List or Set_Attribute_List response has a status not equal to 0.
16#0B	The object is already in the mode or state which has been requested by the service
16#0C	The object is not able to perform the requested service in the current mode or state
16#0D	It has been tried to create an instance of an object which already exists.
16#0E	It has been tried to change an non-modifiable attribute.
16#0F	A check of permissions or privileges failed.
16#10	The current mode or state of the device prevents the execution of the requested service.
16#11	The data to be transmitted in the response buffer requires more space than the size of the allocated response buffer.
16#12	The service specified an operation that is going to fragment a primitive data value, i.e., half a REAL data type.
16#13	The service did not supply all required data to perform the specified operation.
16#14	An unsupported attribute has been specified in the request.
16#15	More data than was expected was supplied by the service.
16#16	The specified object does not exist in the device.
16#17	Fragmentation sequence for this service is not currently active for this data.
16#18	The attribute data of this object has not been saved prior to the requested service.
16#19	The attribute data of this object could not be saved due to a failure during the storage attempt.
16#1A	The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service.
16#1B	The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to abort the service.
16#1C	The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behavior.
16#1D	The service returns the list of attributes containing status information for invalid attributes.
16#1E	An embedded service caused an error.
16#1F	A vendor specific error has occurred. This error should only occur when none of the other general error codes can be applied correctly.
16#20	A parameter which was associated with the request was invalid. The parameter does not meet the requirements of the CIP specification and/or the requirements defined in the specification of an application object.
16#21	An attempt was made to write to a write-once medium for the second time, or to modify a value that cannot be changed after being established once.
16#22	An invalid reply is received. Possible causes can for instance be among others a reply service code not matching the request service code or a reply message shorter than the expectable minimum size.
16#25	The key segment (i.e., the first segment in the path) does not match the destination module. More information about which part of the key check failed can be derived from the object specific status.
16#26	Path cannot be routed to an object due to lacking information or too much routing data has been included.
16#27	It has been attempted to set an attribute which may not be set in the current situation.

Code	
16#28	The Member ID specified in the request is not available within the specified class/ instance or attribute.
16#29	A request to modify a member which cannot be modified has occurred.

Tab. 8-18: CIP general error code

8.4.6 Advanced error codes for connection manager

This section introduces the extended error codes for the Connection Manager.

Code	
16#0100	Connection already in use
16#0103	Transport type not supported
16#0106	More than one guy configuring
16#0107	Trying to close inactive connection
16#0108	Unsupported connection type
16#0109	Connection size mismatch
16#0110	Connection unconfigured
16#0111	Unsupportable RPI
16#0113	Connection Manager out of connections
16#0114	Vendor ID or Product Code mismatch
16#0115	Product Type mismatch
16#0116	Revision mismatch
16#0117	Nonexistent instance number
16#0118	Bad config instance number
16#0119	No controlling connection opened
16#011A	Application out of connections
16#0203	Using a timed out connection
16#0204	Unconnected Send timed out
16#0205	Unconnected Send parameter error
16#0301	No buffer memory available
16#0302	Insufficient bandwidth left
16#0303	Out of gen screeners
16#0304	Not configured to send RT data
16#0305	Signature does not match signature store in CCM
16#0306	CCM is not responding to request
16#0311	Nonexistent port
16#0312	Invalid link address in path
16#0315	Invalid segment in path
16#0316	Path & conn not equal in close

Code	
16#0317	Net segment not present or bad
16#0318	Link address to self invalid
16#0319	Resources in secondary unavailable
16#031D	Redundant connection mismatch
16#0320	Vendor specific: read write access fail
16#2105	Vendor specific: Access beyond end of the requested tag
16#2107	Vendor specific: Data type used in request does not match target tag's data type

Tab. 8-19: Advanced error codes for connection manager

9 RIL_Ethercat.library

9.1 Introduction and overview

The RIL_Ethercat.library is the interface between the PLC programming environment and the EtherCAT master.

Areas of application and function blocks of RIL_EtherCAT library

Area of application (folder in library)	
Function blocks	**Description**
Acyclic communication/CoE (Acyclic Communication/CoE)	
IL_ECATCoeReadSdo, page 258	The function block IL_ECATCoeReadSdo can be used to acyclically retrieve a Service Data Object (SDO) from an EtherCAT slave device.
IL_ECATCoeWriteSdo, page 261	The function block IL_ECATCoeReadSdo can be used to acyclically write a Service Data Object (SDO) from an EtherCAT slave device.
Acyclic communication/SoE (Acyclic Communication/SoE)	
IL_ECATSoeRead, page 263	The function block IL_ECATSoeRead can be used to acyclically read a parameter from an EtherCAT slave device.
IL_ECATSoeWrite, page 267	The function block IL_ECATSoeWrite can be used to acyclically write a parameter from an EtherCAT slave device.
IL_ECATSoeCommand, page 269	The function block IL_ECATSoeCommand can be used to acyclically execute SoE commands of an EtherCAT slave device.
Diagnostics	
IL_ECATGetNumConfigSlaves, page 271	The function block IL_ECATGetNumConfigSlaves returns the number of configured slaves in the EtherCAT master.
IL_ECATGetNumOnlineSlaves, page 272	The function block IL_ECATGetNumOnlineSlaves returns the number of currently connected slaves at the EtherCAT master.
IL_ECATGetConfigEthercatAddrList, page 273	The function block IL_ECATGetConfigEthercatAddrList can be used to read the EtherCAT addresses of the configured slave devices of an EtherCAT master.
IL_ECATGetOnlineEthercatAddrList, page 275	The function block IL_ECATGetOnlineEthercatAddrList can be used to read the EtherCAT addresses of the currently connected slave devices of an EtherCAT master.
IL_ECATGetSlaveConfigInfo, page 276	The function block IL_ECATGetSlaveConfigInfo can be used to retrieve information on a configured slave from the EtherCAT master.
IL_ECATGetSlaveOnlineInfo, page 278	The function block IL_ECATGetSlaveOnlineInfo can be used to retrieve information on a currently connected slave from the EtherCAT master.
IL_ECATMasterState, page 280	The function block IL_ECATMasterState provides a summary of diagnostic information on the EtherCAT master.
IL_ECATMasterBusStatistic, page 281	The function block IL_ECATMasterBusStatistic provides statistic diagnostic information on the bus communication of the EtherCAT master.
IL_ECATRemoteSlaveState, page 283	The function block IL_ECATRemoteSlaveState provides diagnostic status information on an EtherCAT slave device.
IL_ECATRemoteSlaveSetErrorCounterPeriod, page 285	The function block IL_ECATRemoteSlaveSetErrorCounterPeriod enables a master function to cyclically collect the error counters of all slave devices.

IL_ECATRemoteSlaveGetErrorCounterPeriod, page 286	The function block IL_ECATRemoteSlaveGetErrorCounterPeriod reads the current interval of the EtherCAT master to collect the error counters of the slave devices.
IL_ECATRemoteSlaveGetErrorCounter, page 288	The function block IL_ECATRemoteSlaveGetErrorCounter reads out the collected error counters of a slave device from the master.
IL_ECATRemoteSlaveClearErrorCounter, page 289	The function block IL_ECATRemoteSlaveClearErrorCounter deletes all error counters in all slave devices.
IL_ECATCoeReadDiagHistory, page 290	The function block IL_ECATCoeReadDiagHistory can be used to acyclically retrieve the CoE "Diagnosis History" object (index 16#10F3) from an EtherCAT slave device.
State machine	
IL_ECATMasterGetBusState, page 295	The function block IL_ECATMasterGetBusState returns the current bus state of the EtherCAT master.
IL_ECATMasterSetBusState, page 296	The function block IL_ECATMasterSetBusState can be used to request an EtherCAT bus state at the EtherCAT master.
IL_ECATRemoteSlaveGetDevState, page 298	The function block IL_ECATRemoteSlaveGetDevState returns the current EtherCAT state of a slave.
IL_ECATRemoteSlaveSetDevState, page 299	The function block IL_ECATRemoteSlaveSetDevState can be used to request an EtherCAT state at a slave device.
Auxiliary functions (utilities)	
IL_ECATSoeIdnCoding, page 301	The function IL_ECATSoeIdnCoding summarizes the following individual elements in one parameter identification number: Parameter type, parameter block and data block number.
IL_ECATSoeAttributeDecoding, page 301	The function IL_ECATSoeAttributeDecoding decodes the SoE parameter "Attribute".
IL_ECATMasterStateDecoding, page 302	The function IL_ECATMasterStateDecoding decodes the master "State" of the function block IL_ECATMasterState.
IL_ECATSlaveStateDecoding, page 302	The function IL_ECATSlaveStateDecoding decodes the slave "State" of the function block IL_ECATRemoteSlaveState.

Tab. 9-1: RIL_Ethercat.library

EtherCAT® is a registered trademark and patented technology licensed by Beckhoff Automation GmbH, Germany.

9.2 Acyclic communication CoE (Acyclic Communication/CoE)

9.2.1 IL_ECATCoeReadSdo

Brief description The function block IL_ECATCoeReadSdo can be used to acyclically retrieve a Service Data Object (SDO) from an EtherCAT slave device.

Interface description

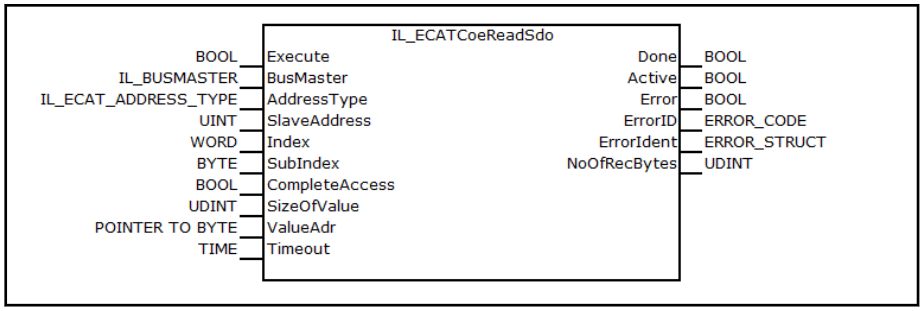


Fig. 9-1: IL_ECATCoeReadSdo function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
	Index	WORD	Index of the object
	SubIndex	BYTE	SubIndex of the object
	CompleteAccess	BOOL	The complete parameter can be read with TRUE (all SubIndices)
	SizeOfValue	UDINT	Size of the buffer or the variable in bytes to receive data (SIZEOF(Value))
	ValueAdr	POINTER TO BYTE	Address of the buffer or the variable to receive data (ADR(Value))
	Timeout	TIME	Timeout monitoring
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NoOfRecBytes	UDINT	Number of bytes copied to the "ValueAdr" buffer

Tab. 9-2: IL_ECATCoeReadSdo interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at “Execute”
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”

Name	Data type	Min.	Max.	Default	Effective
Index	WORD	0	65535	0	Rising edge at “Execute”
SubIndex	BYTE	0	255 [1)]	0	Rising edge at “Execute”
CompleteAccess	BOOL	-	-	FALSE	Rising edge at “Execute”
SizeOfValue	UDINT	0	N. def.	0	Continuous
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Continuous
Timeout	TIME	T#1ms	N. def.	T#3s	Rising edge at “Execute”

Tab. 9-3: Min./max. and default values of the inputs IL_ECATCoeReadSdo

Functional description

The function block IL_ECATCoeReadSdo can be used to acyclically retrieve a Service-Data-Object (SDO) from an EtherCAT slave device.

It is required that the slave device supports the mailbox protocol "CAN application protocol over EtherCAT" (CoE). Mailbox communication is only possible in the EtherCAT states PreOP, SafeOP and OP.

The object to be read out is addressed with "Index" and "SubIndex".

All SubIndices of an object can be read with one call if the setting "CompleteAccess" = TRUE. The start SubIndex is set at "SubIndex":

- SubIndex = 0 → All SubIndices incl. SubIndex 0 (start SubIndex 0)
- SubIndex = 1 → All SubIndices without SubIndex 0 (start SubIndex 1)



"CompleteAccess" is not supported by all slaves or can also be object-specific. If "CompleteAccess" is not supported, the following SDO error codes can result:

- 16#F0ED0047: Unsupported access (CoE abort code 0x06010000)
- 16#F0EE0005: CompleteAccess not supported (CoE abort code 0x06010004)

Implementation in ST

Example:

```
VAR
   fbECATCoeReadSdo : IL_ECATCoeReadSdo;
   boExecute : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   wIndex : WORD := 16#1018; // Identity Object
   bySubIndex : BYTE := 1;   // Vendor ID
   udiVendorId : UDINT;
END_VAR

// Example to read the 'VendorId' from the Identity object
fbECATCoeReadSdo(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   Index:= wIndex,
   SubIndex:= bySubIndex,
   CompleteAccess:= ,
   SizeOfValue:= SIZEOF(udiVendorId),
   ValueAdr:= ADR(udiVendorId),
   Timeout:= ,
   Done=> ,
   Active=> ,
```

1) *For "CompleteAccess" = TRUE, the max. value = 1*

```
Error=> ,
ErrorID=> ,
ErrorIdent=> ,
NoOfRecBytes=> );
```

Error description Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	16#00000001	Input "ValueAdr" invalid: Null pointer
INPUT_INVALID_ERROR	16#F0EC8000	16#00000002	Input "SizeOfValue" invalid: Value "0" not permitted
INPUT_INVALID_ERROR	16#F0EC8000	16#00000003	Input "SubIndex" invalid: Only "0" or "1" permitted for "CompleteAccess"=TRUE
INPUT_INVALID_ERROR	16#F0EC8000	16#00000004	Input "Timeout" invalid: Value "T#0ms" not permitted

Also refer to the general error codes “EtherCAT error table“.

9.2.2 IL_ECATCoeWriteSdo

Brief description The function block IL_ECATCoeReadSdo can be used to acyclically write a Service Data Object (SDO) from an EtherCAT slave device.

Interface description

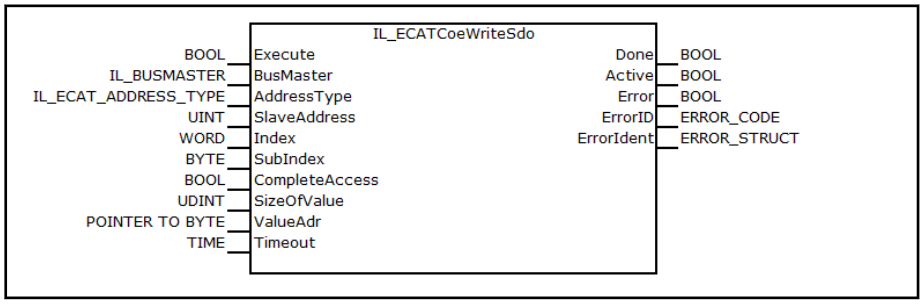


Fig. 9-2: Functional description IL_ECATCoeWriteSdo

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
	Index	WORD	Index of the object
	SubIndex	BYTE	SubIndex of the object
	CompleteAccess	BOOL	The complete parameter can be written with TRUE (all SubIndices)
	SizeOfValue	UDINT	Number of bytes at "ValueAdr" to be written (SIZEOF(Value))
	ValueAdr	POINTER TO BYTE	Address of the buffer or the variable used to write data (ADR(Value))
	Timeout	TIME	Timeout monitoring
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid

	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error

Tab. 9-4: IL_ECATCoeWriteSdo interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
AddressType	IL_ECAT_AD-DRESS_TYPE	ECAT_AUTO-INC_ADDRESS	ECAT_ETHER-CAT_ADDRESS	ECAT_ETHER-CAT_AD-DRESS	Rising edge at “Execute”
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”
Index	WORD	0	65535	0	Rising edge at “Execute”
SubIndex	BYTE	0	255 [2)]	0	Rising edge at “Execute”
CompleteAc-cess	BOOL	-	-	FALSE	Rising edge at “Execute”
SizeOfValue	UDINT	0	N. def.	0	Continuous
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Continuous
Timeout	TIME	T#1ms	N. def.	T#3s	Rising edge at “Execute”

Tab. 9-5: Min./max. and default values of the inputs IL_ECATCoeWriteSdo

Functional description

The function block IL_ECATCoeReadSdo can be used to acyclically write a Service Data Object (SDO) from an EtherCAT slave device.

It is required that the slave device supports the mailbox protocol "CAN application protocol over EtherCAT" (CoE). Mailbox communication is only possible in the EtherCAT states PreOP, SafeOP and OP.

The object to be written is addressed with "Index" and "SubIndex".

All SubIndices of an object can be written with one call if the setting "CompleteAccess" = TRUE. The start SubIndex is set at "SubIndex":

- SubIndex = 0 → All SubIndices incl. SubIndex 0 (start SubIndex 0)
- SubIndex = 1 → All SubIndices without SubIndex 0 (start SubIndex 1)

☞ "CompleteAccess" is not supported by all slaves or can also be object-specific. If "CompleteAccess" is not supported, the following SDO error codes can result:

- 16#F0ED0047: Unsupported access (CoE abort code 0x06010000)
- 16#F0EE0005: CompleteAccess not supported (CoE abort code 0x06010004)

2) *For "CompleteAccess" = TRUE, the max. value = 1*

Implementation in ST

Example:

```
VAR
   fbECATCoeWriteSdo : IL_ECATCoeWriteSdo;
   boExecute : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   wIndex : WORD := 16#4000; // device specific Index/SubIndex
   bySubIndex : BYTE := 3;   // ... which is writeable!
   byWriteValue : BYTE := 16#FF;
END_VAR

// Example to write a device specific byte value
fbECATCoeWriteSdo(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   Index:= wIndex,
   SubIndex:= bySubIndex,
   CompleteAccess:= ,
   SizeOfValue:= SIZEOF(byWriteValue),
   ValueAdr:= ADR(byWriteValue),
   Timeout:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> );
```

Error description

Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	16#00000001	Input "ValueAdr" invalid: Null pointer
INPUT_INVALID_ERROR	16#F0EC8000	16#00000002	Input "SizeOfValue" invalid: Value "0" not permitted
INPUT_INVALID_ERROR	16#F0EC8000	16#00000003	Input "SubIndex" invalid: Only "0" or "1" permitted for "CompleteAccess"=TRUE
INPUT_INVALID_ERROR	16#F0EC8000	16#00000004	Input "Timeout" invalid: Value "t#0ms" not permitted

Also refer to the general error codes “EtherCAT error table“.

9.3 Acyclic communication/SoE (Acyclic Communication/SoE)

9.3.1 IL_ECATSoeRead

Brief description

The function block IL_ECATSoeRead can be used to acyclically read a parameter from an EtherCAT slave device.

Interface description

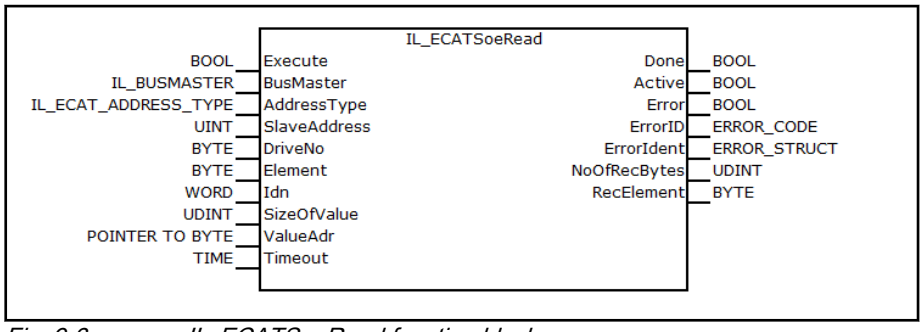


Fig. 9-3: IL_ECATSoeRead function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
	DriveNo	BYTE	Drive number (for single-axis device "0")
	Element	BYTE	Element number of the parameter, refer to IL_ECAT_SOE_ELEMENT
	Idn	WORD	Identification number of the parameter (refer to chapter 9.6.1 "IL_ECATSoeIdnCoding" on page 301)
	SizeOfValue	UDINT	Size of the buffer or the variable in bytes to receive data (SIZEOF(Value))
	ValueAdr	POINTER TO BYTE	Address of the buffer or the variable to receive data (ADR(Value))
	Timeout	TIME	Timeout monitoring
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NoOfRecBytes	UDINT	Number of bytes copied to the "ValueAdr" buffer
	RecElement	BYTE	For receiving elements, refer to IL_ECAT_SOE_ELEMENT (if more than one bit was set at the "Element" input)

Tab. 9-6: IL_ECATSoeRead interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at “Execute”
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”
DriveNo	BYTE	0	7	0	Rising edge at “Execute”
Element	BYTE	16#01	16#FF	SOE_ELEMENT_OPDATA	Rising edge at “Execute”
Idn	WORD	0	N. def.	0	Rising edge at “Execute”

Name	Data type	Min.	Max.	Default	Effective
SizeOfValue	UDINT	0	N. def.	0	Continuous
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Continuous
Timeout	TIME	T#1ms	N. def.	T#3s	Rising edge at “Execute”

Tab. 9-7: Min./max. and default values of the inputs IL_ECATSoeRead

Functional description

The function block IL_ECATSoeRead can be used to acyclically read a parameter from an EtherCAT slave device.

It is required that the slave device supports the mailbox protocol "Servo drive profile over EtherCAT" (SoE). Mailbox communication is only possible in the EtherCAT states PreOP, SafeOP and OP.

The parameter to be retrieved is addressed with "Idn", "Element" and "DriveNo". The function chapter 9.6.1 "IL_ECATSoeIdnCoding" on page 301 is recommended for Idn coding.

The enumeration type "IL_ECAT_SOE_ELEMENT" is recommended for the Element input. Multiple bits can also be set. Thus, multiple elements can be retrieved with one acyclic request.

For example Sercos attribute and data:

```
fbECATSoeRead.Element := SOE_ELEMENT_ATTRIBUTE OR SOE_ELEMENT_OPDATA;
```

In the response data, the values of the requested elements are entered in the element bit sequence. The elements transferred can be checked at the "RecElement" output.

☞ Reading multiple elements with one acyclic request is an optional functionality in the slave devices. If this is not supported by the slave, the slave only responds with the data of the first element bit set (refer to the "RecElement" output). This is not an error, i.e. the function block does not report an error. The requested elements have to be read out one after the other.

☞ If a list parameter is read, the read out data shows the current length of the list at the bits 0 and 1, the maximum length of the list at the bytes 2 and 3 and data of the list parameter starting from byte 4.

Implementation in ST

Example:

```
(*=================================================================*)

// Example A : read a UINT Parameter (S-0-0002)

VAR
   fbECATSoeRead  : IL_ECATSoeRead;
   boExecute      : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   uiCycleTime    : UINT;
END_VAR

fbECATSoeRead(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   DriveNo:= ,
   Element:= ,
   Idn:= IL_ECATSoeIdnCoding(SOE_S_PARAM,0,2), // ^= S-0-0002
```

```
   SizeOfValue:= SIZEOF(uiCycleTime),
   ValueAdr:= ADR(uiCycleTime),
   Timeout:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   NoOfRecBytes=> );

============================================================

// Example B : read a STRING Parameter (S-0-0095)

//-----------------------------------------------------------
// Structure for SoE String param. (= List with var. length)
TYPE ST_SOE_STRING :
STRUCT
   ActLength    : UINT; // actual length without 0-ending
   MaxLength    : UINT; // max length without 0-ending
   Text         : STRING(255); // text value
END_STRUCT
END_TYPE
//-----------------------------------------------------------

VAR
   fbECATSoeRead  : IL_ECATSoeRead;
   boExecute      : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   rDiagnosis     : ST_SOE_STRING;
END_VAR

fbECATSoeRead(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   DriveNo:= ,
   Element:= ,
   Idn:= IL_ECATSoeIdnCoding(SOE_S_PARAM,0,95), // ^= S-0-0095
   SizeOfValue:= SIZEOF(rDiagnosis),
   ValueAdr:= ADR(rDiagnosis),
   Timeout:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   NoOfRecBytes=> );

============================================================

// Example C : read an array (S-0-0017)

//-----------------------------------------------------------
// Structure for a SoE List parameter (with WORD elements)
TYPE ST_SOE_WORD_LIST :
STRUCT
   ActLength    : UINT; // actual length
   MaxLength    : UINT; // max length
   awIDN        : ARRAY[0..9999] OF WORD; // List elements
END_STRUCT
END_TYPE
//-----------------------------------------------------------

VAR
   fbECATSoeRead  : IL_ECATSoeRead;
   boExecute      : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   rValue         : ST_SOE_WORD_LIST;
END_VAR

fbECATSoeRead(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   DriveNo:= ,
   Element:= ,
   Idn:= IL_ECATSoeIdnCoding(SOE_S_PARAM,0,17), // ^= S-0-0017
   SizeOfValue:= SIZEOF(rValue),
```

```
ValueAdr:= ADR(rValue),
Timeout:= ,
Done=> ,
Active=> ,
Error=> ,
ErrorID=> ,
ErrorIdent=> ,
NoOfRecBytes=> );
```

```
=============================================================
```

Error description Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	16#00000001	Input "ValueAdr" invalid: Null pointer
INPUT_INVALID_ERROR	16#F0EC8000	16#00000002	Input "SizeOfValue" invalid: Value "0" not permitted
INPUT_INVALID_ERROR	16#F0EC8000	16#00000004	Input "Timeout" invalid: Value "T#0ms" not permitted

Also refer to the general error codes “EtherCAT error table“.

9.3.2 IL_ECATSoeWrite

Brief description The function blockIL_ECATSoeWrite can be used to acyclically write a parameter from an EtherCAT slave device.

Interface description

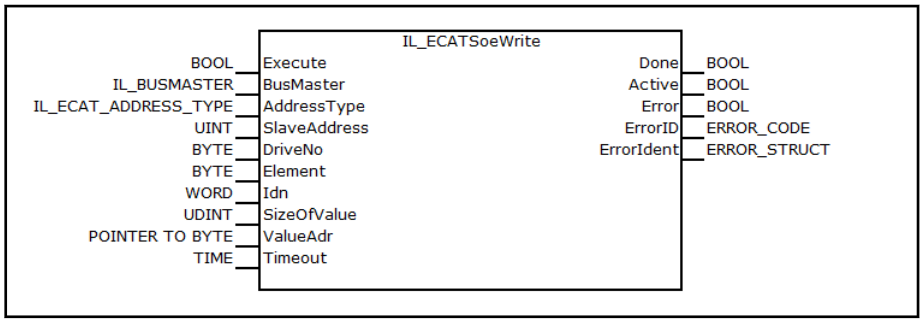


Fig. 9-4: IL_ECATSoeWrite function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
	DriveNo	BYTE	Drive number (for single-axis device "0")
	Element	BYTE	Element number of the parameter, refer to IL_ECAT_SOE_ELEMENT
	Idn	WORD	Identification number of the parameter (refer to chapter 9.6.1 "IL_ECATSoeIdnCoding" on page 301)
	SizeOfValue	UDINT	Number of bytes at "ValueAdr" to be written (SIZEOF(Value))
	ValueAdr	POINTER TO BYTE	Address of the buffer or the variable used to write data (ADR(Value))

I/O type	Name	Data type	Description
	Timeout	TIME	Timeout monitoring
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error

Tab. 9-8: IL_ECATSoeWrite interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
AddressType	IL_ECAT_AD-DRESS_TYPE	ECAT_AUTO-INC_ADDRESS	ECAT_ETHER-CAT_ADDRESS	ECAT_ETHER-CAT_AD-DRESS	Rising edge at “Execute”
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”
DriveNo	BYTE	0	7	0	Rising edge at “Execute”
Element	BYTE	16#01	16#FF	SOE_ELE-MENT_OPDA-TA	Rising edge at “Execute”
Idn	WORD	0	N. def.	0	Rising edge at “Execute”
SizeOfValue	UDINT	0	N. def.	0	Continuous
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Continuous
Timeout	TIME	T#1ms	N. def.	T#3s	Rising edge at “Execute”

Tab. 9-9: Min./max. and default values of the inputs IL_ECATSoeWrite

Functional description

The function blockIL_ECATSoeWrite can be used to acyclically write a parameter from an EtherCAT slave device.

It is required that the slave device supports the mailbox protocol "Servo drive profile over EtherCAT" (SoE). Mailbox communication is only possible in the EtherCAT states PreOP, SafeOP and OP.

The parameter to be written is addressed with "Idn", "Element" and "DriveNo". The function chapter 9.6.1 "IL_ECATSoeIdnCoding" on page 301 is recommended for Idn coding.



To write a list parameter, 4 bytes have to be supplemented at the beginning of the data to be written: The bytes 0 and 1 have to contain the current length and the bytes 2 and 3 are reserved for the maximum data length (this maximum length does not have to be set).

Implementation in ST

Example:

```
VAR
   fbECATSoeWrite : IL_ECATSoeWrite;
```

```
   boExecute : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   uiWriteValue : UINT := 16#1234;
END_VAR

// Example to write a '2 Byte Parameter' (here: S-0-0000)
fbECATSoeWrite(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   DriveNo:= ,
   Element:= ,
   Idn:= IL_ECATSoeIdnCoding(SOE_S_PARAM,0,0), // ^= S-0-0000
   SizeOfValue:= SIZEOF(uiWriteValue),
   ValueAdr:= ADR(uiWriteValue),
   Timeout:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> );
```

Error description Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	16#00000001	Input "ValueAdr" invalid: Null pointer
INPUT_INVALID_ERROR	16#F0EC8000	16#00000002	Input "SizeOfValue" invalid: Value "0" not permitted
INPUT_INVALID_ERROR	16#F0EC8000	16#00000004	Input "Timeout" invalid: Value "T#0ms" not permitted

Also refer to the general error codes “EtherCAT error table“.

9.3.3 IL_ECATSoeCommand

Brief description The function block IL_ECATSoeCommand can be used to acyclically execute SoE commands of an EtherCAT slave device.

Interface description

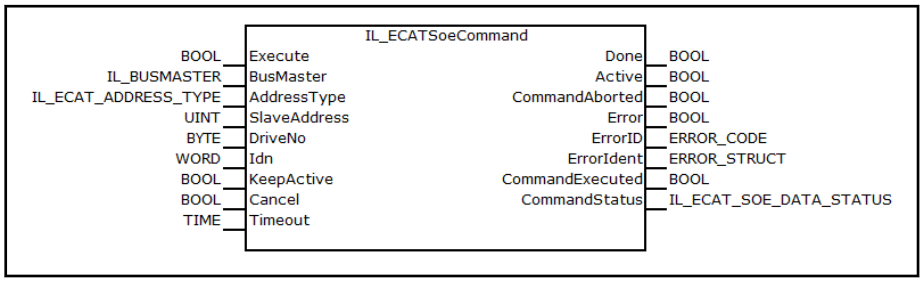


Fig. 9-5: IL_ECATSoeCommand function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
	DriveNo	BYTE	Drive number (for single-axis device "0")
	Idn	WORD	Identification number of the command parameter (refer to chapter 9.6.1 "IL_ECATSoeIdnCoding" on page 301)

I/O type	Name	Data type	Description
	KeepActive	BOOL	Keeps the command active after processing (state-controlled)
	Cancel	BOOL	Aborts the running command (at a rising edge)
	Timeout	TIME	Timeout monitoring of internally used acyclic services
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	CommandAborted	BOOL	Command successfully aborted and reset
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	CommandExecuted	BOOL	Command successfully completed
	CommandStatus	IL_ECAT_SOE_DATA_STATUS	Status of the command parameter

Tab. 9-10: IL_ECATSoeCommand interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at “Execute”
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”
DriveNo	BYTE	0	7	0	Rising edge at “Execute”
Idn	WORD	0	N. def.	0	Rising edge at “Execute”
KeepActive	BOOL	-	-	FALSE	Continuous
Cancel	BOOL	-	-	FALSE	Continuous
Timeout	TIME	T#1ms	N. def.	T#3s	Rising edge at “Execute”

Tab. 9-11: Min./max. and default values of the inputs IL_ECATSoeCommand

Functional description

The function block IL_ECATSoeCommand can be used to acyclically execute SoE commands of an EtherCAT slave device.

It is required that the slave device supports the mailbox protocol "Servo drive profile over EtherCAT" (SoE). Mailbox communication is only possible in the EtherCAT states PreOP, SafeOP and OP.

The command parameter to be executed, is addressed with "Idn" and "DriveNo". The function chapter 9.6.1 "IL_ECATSoeIdnCoding" on page 301 is recommended for Idn coding.

The command is automatically reset after processing. Depending on the respective command, its execution can be canceled at any time via the "Cancel" input.

The "KeepActive" input allows the command to remain active after a successful execution. If "KeepActive" is disabled later on, the command is automatically reset.

Implementation in ST

Example:

```
VAR
   fbECATSoeCommand : IL_ECATSoeCommand;
   boExecute : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
END_VAR

// Example to execute the Reset command (S-0-0099)
fbECATSoeCommand(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   DriveNo:= ,
   Idn:= IL_ECATSoeIdnCoding(SOE_S_PARAM,0,99), // ^= S-0-0099
   KeepActive:= ,
   Cancel:= ,
   Timeout:= ,
   Done=> ,
   Active=> ,
   CommandAborted=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   CommandExecuted=> ,
   CommandStatus=> );
```

Error description

Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	Internal	Input "Idn" is no command parameter
DEVICE_ERROR	16#F0EC8010	Internal	Invalid command data status, refer to the output "CommandStatus".

Also refer to the general error codes “EtherCAT error table“.

9.4 Diagnostics

9.4.1 IL_ECATGetNumConfigSlaves

Brief description

The function block IL_ECATGetNumConfigSlaves returns the number of configured slaves in the EtherCAT master.

Interface description

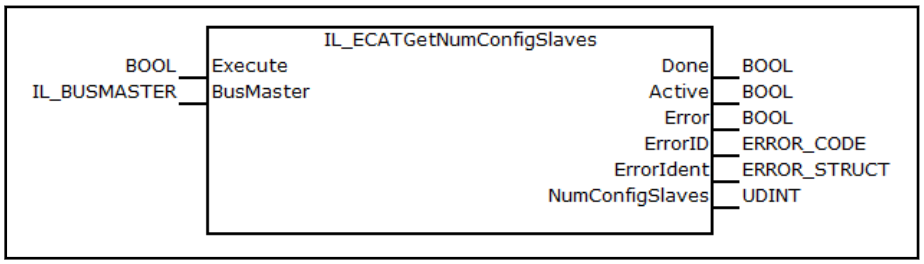


Fig. 9-6: IL_ECATGetNumConfigSlaves function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NumConfig-Slaves	UDINT	Number of configured slaves in the EtherCAT master

Tab. 9-12: IL_ECATGetNumConfigSlaves interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”

Tab. 9-13: Min./max. and default values of the inputs IL_ECATGetNumConfig-Slaves

Functional description

The function block IL_ECATGetNumConfigSlaves returns the number of configured slaves in the EtherCAT master.

Implementation in ST

Example:

```
VAR
   fbECATGetNumConfigSlaves: IL_ECATGetNumConfigSlaves;
   boExecute : BOOL := FALSE;
   udiNumConfigSlaves : UDINT;
END_VAR

fbECATGetNumConfigSlaves(
   Execute:= boExecute,
   BusMaster:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   NumConfigSlaves=> udiNumConfigSlaves );
```

Error description

No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.2 IL_ECATGetNumOnlineSlaves

Brief description

The function block IL_ECATGetNumOnlineSlaves returns the number of currently connected slaves at the EtherCAT master.

Interface description

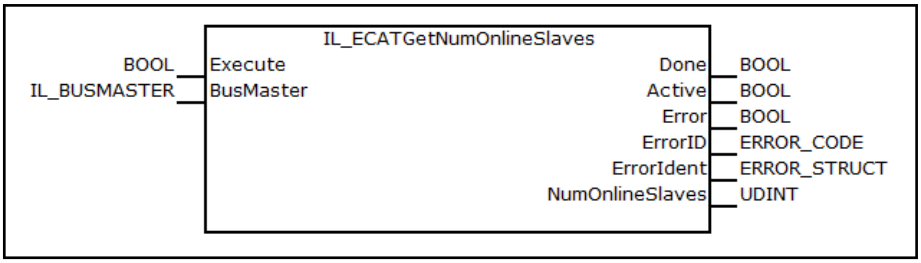


Fig. 9-7: IL_ECATGetNumOnlineSlaves function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NumOnlineSlaves	UDINT	Number of currently connected slaves at the EtherCAT master

Tab. 9-14: IL_ECATGetNumOnlineSlaves interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”

Tab. 9-15: Min./max. and default values of the inputs IL_ECATGetNumOnline-Slaves

Functional description

The function block IL_ECATGetNumOnlineSlaves returns the number of currently connected slaves at the EtherCAT master.

Implementation in ST

Example:

```
VAR
   fbECATGetNumOnlineSlaves: IL_ECATGetNumOnlineSlaves;
   boExecute : BOOL := FALSE;
   udiNumOnlineSlaves : UDINT;
END_VAR

fbECATGetNumOnlineSlaves(
   Execute:= boExecute,
   BusMaster:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   NumOnlineSlaves=> udiNumOnlineSlaves );
```

Error description

No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.3 IL_ECATGetConfigEthercatAddrList

Brief description

The function block IL_ECATGetConfigEthercatAddrList can be used to read the EtherCAT addresses of the configured slave devices of an EtherCAT master.

Interface description

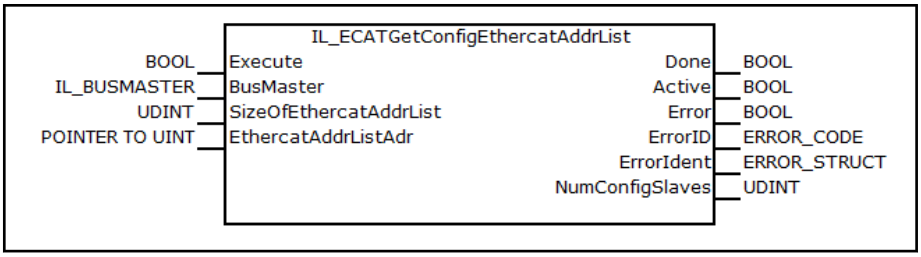


Fig. 9-8: IL_ECATGetConfigEthercatAddrList function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	SizeOfEthercatAddrList	UDINT	Size of a data range in which the EtherCAT addresses are written (SIZEOF(EthercatAddrList))
	EthercatAddrListAdr	POINTER TO UINT	Pointer to a data area to which the EtherCAT addresses are written (ADR(EthercatAddrList))
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NumConfigSlaves	UDINT	Number of configured slaves (= valid entries in "EthercatAddrList")

Tab. 9-16: IL_ECATGetConfigEthercatAddrList interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SizeOfEthercatAddrList	UDINT	0	N. def.	0	Rising edge at “Execute”
EthercatAddrListAdr	POINTER TO UINT	N. def.	N. def.	0	Continuous

Tab. 9-17: Min./max. and default values of the inputs IL_ECATGetConfigEthercatAddrList

Functional description

The function block IL_ECATGetConfigEthercatAddrList can be used to read the EtherCAT addresses of the configured slave devices of an EtherCAT master.

Implementation in ST

Example:

```
VAR
   fbECATGetConfigEthercatAddrList : IL_ECATGetConfigEthercatAddrList;
   boExecute : BOOL := FALSE;
   // array with max number of configured slaves, here: '128'
   arEthercatAddrList : ARRAY [0..127] OF UINT;
   udiNumConfigSlaves : UDINT;
```

```
END_VAR

// Example to read the configured EtherCAT Addresses
fbECATGetConfigEthercatAddrList(
   Execute:= boExecute,
   BusMaster:= ,
   SizeOfEthercatAddrList:= SIZEOF(arEthercatAddrList),
   EthercatAddrListAdr:= ADR(arEthercatAddrList),
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   NumConfigSlaves=> udiNumConfigSlaves);
```

Error description

Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	Internal	Input "EthercatAddrListAdr" invalid: Null pointer
INPUT_INVALID_ERROR	16#F0EC8001	Internal	Input "SizeOfEthercatAddrListAdr" invalid: Data range is too small

Also refer to the general error codes “Error codes error table“.

9.4.4 IL_ECATGetOnlineEthercatAddrList

Brief description

The function block IL_ECATGetOnlineEthercatAddrList can be used to read the EtherCAT addresses of the currently connected slave devices of an EtherCAT master.

Interface description

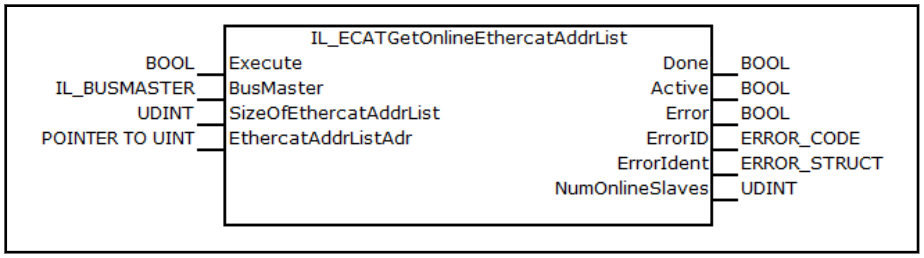


Fig. 9-9: IL_ECATGetOnlineEthercatAddrList function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	SizeOfEthercatAddrList	UDINT	Size of a data range in which the EtherCAT addresses are written (SIZEOF(EthercatAddrList))
	EthercatAddrListAdr	POINTER TO UINT	Pointer to a data area to which the EtherCAT addresses are written (ADR(EthercatAddrList))
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	NumOnlineSlaves	UDINT	Number of currently connected slaves (= valid entries in "EthercatAddrList")

Tab. 9-18: IL_ECATGetOnlineEthercatAddrList interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
SizeOfEther-catAddrList	UDINT	0	N. def.	0	Rising edge at “Execute”
EthercatAddr-ListAdr	POINTER TO UINT	N. def.	N. def.	0	Continuous

Tab. 9-19: Min./max. and default values of the inputs IL_ECATGetOnlineEther-catAddrList

Functional description

The function block IL_ECATGetOnlineEthercatAddrList can be used to read the EtherCAT addresses of the currently connected slave devices of an EtherCAT master.

Implementation in ST

Example:

```
VAR
   fbECATGetOnlineEthercatAddrList : IL_ECATGetOnlineEthercatAddrList;
   boExecute: BOOL := FALSE;
   // array with max number of connected slaves, here '128'
   arEthercatAddrList : ARRAY [0..127] OF UINT;
   udiNumOnlineSlaves : UDINT;
END_VAR

// Example to read the currently connected EtherCAT Addresses
fbECATGetOnlineEthercatAddrList(
   Execute:= boExecute,
   BusMaster:= ,
   SizeOfEthercatAddrList:= SIZEOF(arEthercatAddrList),
   EthercatAddrListAdr:= ADR(arEthercatAddrList),
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   NumOnlineSlaves=> udiNumOnlineSlaves );
```

Error description

Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	Internal	Input "EthercatAddrListAdr" invalid: Null pointer
INPUT_INVALID_ERROR	16#F0EC8001	Internal	Input "SizeOfEthercatAddrListAdr" invalid: Data range is too small

Also refer to the general error codes “EtherCAT error table“.

9.4.5 IL_ECATGetSlaveConfigInfo

Brief description

The function block IL_ECATGetSlaveConfigInfo can be used to retrieve information on a configured slave from the EtherCAT master.

Interface description

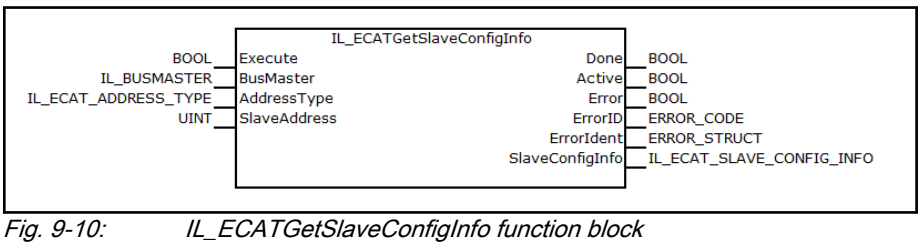


Fig. 9-10: IL_ECATGetSlaveConfigInfo function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the configured AutoInc address (depends on the input "AddressType")
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	SlaveConfigInfo	IL_ECAT_SLAVE_CONFIG_INFO	Information on a configured slave

Tab. 9-20: IL_ECATGetSlaveConfigInfo interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at Execute
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”

Tab. 9-21: Min./max. and default values of the inputs IL_ECATGetSlaveConfigInfo

Functional description

The function block IL_ECATGetSlaveConfigInfo can be used to retrieve information on a configured slave from the EtherCAT master.

☞ For the EtherCAT configuration of the slaves, the AutoInc address has to be assigned consecutively without gaps [0, 0xFFFF (-1), 0xFFFE (-2), …]. Thus, the info data of all configured slaves can be read out with a simple loop, refer to the following programming example.

Implementation in ST

Example:

```
VAR
   boExecute : BOOL := FALSE;
   boDone    : BOOL := FALSE;
   boError   : BOOL := FALSE;
   // arrays with max number of configured slaves, here: '128'
   arEthercatAddrList : ARRAY[0..127] OF UINT;
   arSlaveNameList : ARRAY[0..127] OF STRING(ECAT_MAX_SLAVE_NAME_LEN);
   udiNumConfigSlaves : UDINT;
   uiSlaveIdx : UINT := 0;
   fbECATGetNumConfigSlaves: IL_ECATGetNumConfigSlaves;
   fbECATGetSlaveConfigInfo: IL_ECATGetSlaveConfigInfo;
END_VAR

// Example to read the EthercatAddr and the SlaveName
```

```
// from all configured Slaves

IF (boExecute = TRUE  AND  boDone = FALSE) THEN

   // get number of configured slaves
   fbECATGetNumConfigSlaves(
      Execute:= TRUE,
      NumConfigSlaves=> udiNumConfigSlaves );
   IF (TRUE = fbECATGetNumConfigSlaves.Error) THEN
      boError := TRUE;
   ELSIF (TRUE = fbECATGetNumConfigSlaves.Done) THEN

      // read SlaveConfigData from one slave after the other
      IF (uiSlaveIdx < udiNumConfigSlaves) THEN

         fbECATGetSlaveConfigInfo(
            Execute:= TRUE,
            AddressType:= ECAT_AUTOINC_ADDRESS,
            SlaveAddress:= (0 - uiSlaveIdx) );
         IF (TRUE = fbECATGetSlaveConfigInfo.Error) THEN
            boError := TRUE;
         ELSIF (TRUE = fbECATGetSlaveConfigInfo.Done) THEN
            arEthercatAddrList[uiSlaveIdx] :=
                  fbECATGetSlaveConfigInfo.SlaveConfigInfo.EthercatAddr;
            arSlaveNameList[uiSlaveIdx] :=
                  fbECATGetSlaveConfigInfo.SlaveConfigInfo.SlaveName;
            // next slave
            fbECATGetSlaveConfigInfo(Execute:=FALSE);
            uiSlaveIdx := uiSlaveIdx + 1;
         END_IF

      ELSE // uiSlaveIdx
         boDone := TRUE; // Finished
      END_IF
   END_IF // fbECATGetNumConfigSlaves

ELSIF (boExecute = FALSE) THEN
   boDone := FALSE;
   boError:= FALSE;
   uiSlaveIdx := 0;
   fbECATGetNumConfigSlaves(Execute:=FALSE);
   fbECATGetSlaveConfigInfo(Execute:=FALSE);
END_IF
```

Error description

No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.6 IL_ECATGetSlaveOnlineInfo

Brief description

The function block IL_ECATGetSlaveOnlineInfo can be used to retrieve information on a currently connected slave from the EtherCAT master.

Interface description

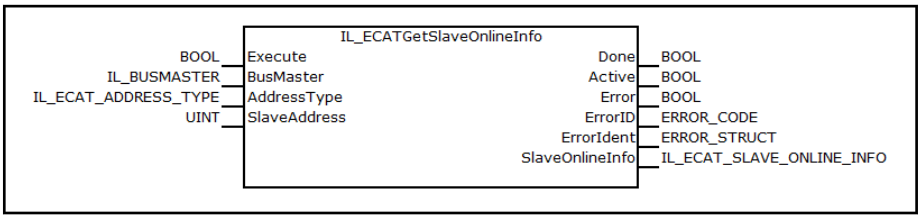


Fig. 9-11: IL_ECATGetSlaveOnlineInfo function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	SlaveOnlineInfo	IL_ECAT_SLAVE_ONLINE_INFO	Information on a currently connected slave

Tab. 9-22: IL_ECATGetSlaveOnlineInfo interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at Execute
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”

Tab. 9-23: Min./max. and default values of the inputs IL_ECATGetSlaveOnlineInfo

Functional description

The function block IL_ECATGetSlaveOnlineInfo can be used to retrieve information on a currently connected slave from the EtherCAT master.

☞ The EtherCAT master automatically executes a bus scan if a change in topology is detected ("DL status event"). The function IL_ECATGetSlaveOnlineInfo retrieves the collected data from the EtherCAT master.

Implementation in ST

Example:

```
VAR
   boExecute : BOOL := FALSE;
   boDone    : BOOL := FALSE;
   boError   : BOOL := FALSE;
   // array with max number of connected slaves, here: '128'
   arSlaveOnlineInfo : ARRAY[0..127] OF IL_ECAT_SLAVE_ONLINE_INFO;
   udiNumOnlineSlaves: UDINT;
   uiSlaveIdx : UINT := 0;
   fbECATGetNumOnlineSlaves : IL_ECATGetNumOnlineSlaves;
   fbECATGetSlaveOnlineInfo : IL_ECATGetSlaveOnlineInfo;
END_VAR

// Example to read all SlaveOnlineInfo data

IF (boExecute = TRUE  AND  boDone = FALSE) THEN

   // get number of currently connected slaves
   fbECATGetNumOnlineSlaves(
      Execute:= TRUE,
      NumOnlineSlaves=> udiNumOnlineSlaves);
   IF (TRUE = fbECATGetNumOnlineSlaves.Error) THEN
      boError := TRUE;
   ELSIF (TRUE = fbECATGetNumOnlineSlaves.Done) THEN

      // read OnlineSlaveInfo from one slave after the other
      IF (uiSlaveIdx < udiNumOnlineSlaves) THEN

         fbECATGetSlaveOnlineInfo(
```

```
                Execute:= TRUE,
                AddressType:= ECAT_AUTOINC_ADDRESS,
                SlaveAddress:= (0 - uiSlaveIdx),
                SlaveOnlineInfo => arSlaveOnlineInfo[uiSlaveIdx] );
             IF (TRUE = fbECATGetSlaveOnlineInfo.Error) THEN
                boError := TRUE;
                // Note: maybe the number of OnlineSlaves has just changed
             ELSIF (TRUE = fbECATGetSlaveOnlineInfo.Done) THEN
                // next slave
                fbECATGetSlaveOnlineInfo(Execute:=FALSE);
                uiSlaveIdx := uiSlaveIdx + 1;
             END_IF

          ELSE // uiSlaveIdx
             boDone := TRUE; // Finished
          END_IF
       END_IF // fbECATGetNumOnlineSlaves

ELSIF (boExecute = FALSE) THEN
   boDone := FALSE;
   boError:= FALSE;
   uiSlaveIdx := 0;
   fbECATGetNumOnlineSlaves(Execute:=FALSE);
   fbECATGetSlaveOnlineInfo(Execute:=FALSE);
END_IF
```

Error description No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.7 IL_ECATMasterState

Brief description The function block IL_ECATMasterState provides a summary of diagnostic information on the EtherCAT master.

Interface description

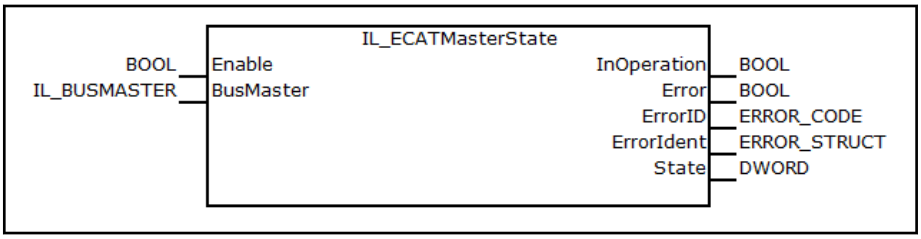


Fig. 9-12: IL_ECATMasterState function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
VAR_OUTPUT	InOperation	BOOL	Cyclic processing state
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	State	DWORD	Diagnostic status information of the EtherCAT master. For details, refer to IL_ECAT_MASTER_STATE

Tab. 9-24: IL_ECATMasterState interface

Name	Data type	Min.	Max.	Default	Effective
Enable	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Enable”

Tab. 9-25: Min./max. and default values of the inputs IIL_ECATMasterState

Functional description

The function block IL_ECATMasterState provides a summary of diagnostic information on the EtherCAT master.

If the output "State" = 0, no diagnostics is present and the master is OK. For a description of the diagnostic bits of the "State" output, refer to the structure IL_ECAT_MASTER_STATE as well as to the auxiliary function IL_ECATMasterStateDecoding.

Implementation in ST

Example:

```
VAR
   fbECATMasterState : IL_ECATMasterState;
   boEnable     : BOOL := FALSE;
   boMasterOK   : BOOL; // FALSE = Master diagnosis available
   rMasterState : IL_ECAT_MASTER_STATE;
   eActBusState : IL_ECAT_STATE;
END_VAR

// Example to read Master State diagnosis (cyclic call)
fbECATMasterState(
   Enable:= boEnable,
   BusMaster:= ,
   InOperation=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   State=> );

IF (TRUE = fbECATMasterState.InOperation) THEN
   // Master OK
   boMasterOK := NOT( DWORD_TO_BOOL(fbECATMasterState.State) );
   // Master diagnosis bits
   rMasterState := IL_ECATMasterStateDecoding( fbECATMasterState.State );
   // Actual Bus State
   IF (FALSE = rMasterState.NotInStateOp) THEN
      eActBusState := ECAT_STATE_OP;
   ELSE
      eActBusState := DWORD_TO_DINT( SHR((fbECATMasterState.State AND 16#0070), 4) );
   END_IF
ELSE
   boMasterOK := FALSE;
   eActBusState := ECAT_STATE_UNKNOWN;
   ; // ToDo error handling
END_IF
```

Error description

No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.8 IL_ECATMasterBusStatistic

Brief description

The function block IL_ECATMasterBusStatistic provides statistic diagnostic information on the bus communication of the EtherCAT master.

Interface description

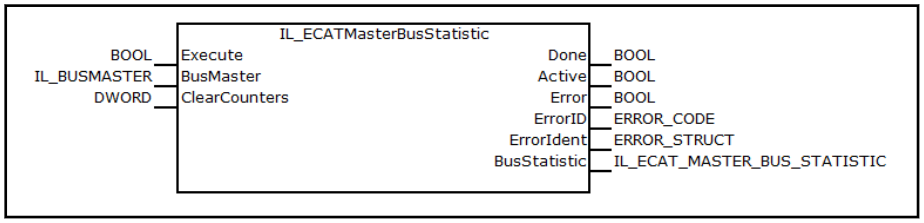


Fig. 9-13: IL_ECATMasterBusStatistic function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	ClearCounters	DWORD	Reset of the frame and the data telegram counter after reading (1 = deletes all counters)
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	BusStatistic	IL_ECAT_MASTER_BUS_STATISTIC	Statistic diagnostic information on the bus communication of the EtherCAT master

Tab. 9-26: IL_ECATMasterBusStatistic interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
ClearCounters	DWORD	0	N. def.	0	Rising edge at “Execute”

Tab. 9-27: Min./max. and default values of the inputs IL_ECATMasterBusStatistic

Functional description

The function block IL_ECATMasterBusStatistic provides statistic diagnostic information on the bus communication of the EtherCAT master.

If the input "ClearCounters" is set to > 0, the following counters are reset after reading the data (bit x of the "ClearCounters" input):

- Bit 0: Clears all counters (ClearCounters = 1)
- Bit 1: Clears (SI7) counter "TxFrames"
- Bit 2: Clears (SI8) counter "RxFrames"
- Bit 3: Clears (SI9) counter "LostFrames"
- Bit 4: Clears (SI10) counter "CyclicFrames"
- Bit 5: Clears (SI11) counter "CyclicDatagrams"
- Bit 6: Clears (SI12) counter "AcyclicFrames"
- Bit 7: Clears (SI13) counter "AcyclicDatagrams"

Implementation in ST

Example:

```
VAR
   fbECATMasterBusStatistic : IL_ECATMasterBusStatistic;
   boExecute : BOOL := FALSE;
   // 'ClearCounters': set value '1' to clear all counters
   dwClearCounters : DWORD := 0;
   rBusStatistic : IL_ECAT_MASTER_BUS_STATISTIC;
END_VAR

// Example to read the Master Bus Statistic
fbECATMasterBusStatistic(
   Execute:= boExecute,
   BusMaster:= ,
```

```
ClearCounters:= dwClearCounters,
Done=> ,
Active=> ,
Error=> ,
ErrorID=> ,
ErrorIdent=> ,
BusStatistic=> rBusStatistic);
```

Error description No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.9 IL_ECATRemoteSlaveState

Brief description The function block IL_ECATRemoteSlaveState provides diagnostic status information on an EtherCAT slave device.

Interface description

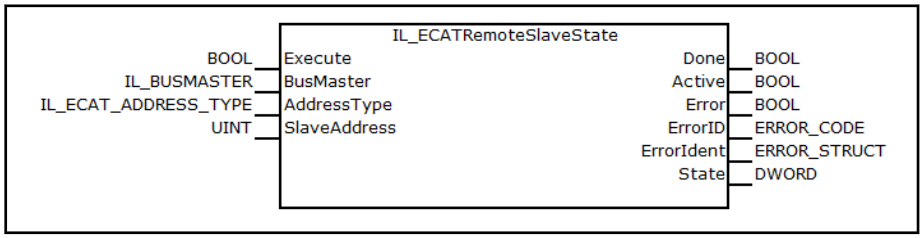


Fig. 9-14: IL_ECATRemoteSlaveState function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	State	DWORD	Diagnostic status information of the slave device; for details, refer to IL_ECAT_SLAVE_STATE

Tab. 9-28: IL_ECATRemoteSlaveState interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at “Execute”
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”

Tab. 9-29: Min./max. and default values of the inputs IL_ECATRemoteSlaveState

Functional description

The function block IL_ECATRemoteSlaveState provides diagnostic status information on an EtherCAT slave device.

If the output "State" = 0, no diagnostics is present and the slave is OK. For a description of the diagnostic bits of the "State" output, refer to the structure IL_ECAT_SLAVE_STATE as well as to the auxiliary function IL_ECATSlaveStateDecoding.

Implementation in ST

Example:

```
VAR CONSTANT
   // max number of configured slaves, here: '128'
   uiMAX_CONFIG_SLAVES : UINT := 128;
END_VAR
VAR
   boEnable           : BOOL := FALSE;
   boInOperation      : BOOL := FALSE;
   boError            : BOOL := FALSE;
   arEthercatAddrList : ARRAY [0..(uiMAX_CONFIG_SLAVES-1)] OF UINT;
   arSlaveOK          : ARRAY [0..(uiMAX_CONFIG_SLAVES-1)] OF BOOL;
   arSlaveState       : ARRAY [0..(uiMAX_CONFIG_SLAVES-1)] OF IL_ECAT_SLAVE_STATE;
   arSlaveDevState    : ARRAY [0..(uiMAX_CONFIG_SLAVES-1)] OF IL_ECAT_STATE;
   udiNumConfigSlaves : UDINT := 0;
   uiSlaveIdx         : UINT := 0;
   pDword             : POINTER TO DWORD;
   fbECATGetConfigEthercatAddrList : IL_ECATGetConfigEthercatAddrList;
   fbECATRemoteSlaveState          : IL_ECATRemoteSlaveState;
END_VAR

// Example to read the RemoteSlaveState from all configured slaves

IF (boEnable = TRUE) THEN

   IF (boInOperation = FALSE) THEN
      (* Init *)

      // get all configured EtherCAT addresses
      fbECATGetConfigEthercatAddrList(
         Execute:= TRUE,
         SizeOfEthercatAddrList:= SIZEOF(arEthercatAddrList),
         EthercatAddrListAdr:= ADR(arEthercatAddrList),
         NumConfigSlaves=> udiNumConfigSlaves );
      IF (fbECATGetConfigEthercatAddrList.Error) THEN
         boError := TRUE;
      ELSIF (fbECATGetConfigEthercatAddrList.Done = TRUE) THEN
         boInOperation := TRUE;
      END_IF

   ELSE //
      (* InOperation *)

      // Hint: you may read the RemoteSlaveStates only if MasterState <> 0

      // read the RemoteSlaveState from one slave after the other
      IF (uiSlaveIdx < udiNumConfigSlaves) THEN

         fbECATRemoteSlaveState(
            Execute:= TRUE,
            AddressType:= ECAT_ETHERCAT_ADDRESS,
            SlaveAddress:= arEthercatAddrList[uiSlaveIdx],
            State=> );

         IF (TRUE = fbECATRemoteSlaveState.Error) THEN
            boError := TRUE;
         ELSIF (TRUE = fbECATRemoteSlaveState.Done) THEN
            // Slave diagnosis
            arSlaveOK[uiSlaveIdx] :=
                  NOT( DWORD_TO_BOOL(fbECATRemoteSlaveState.State) );
            arSlaveState[uiSlaveIdx] :=
                  IL_ECATSlaveStateDecoding( fbECATRemoteSlaveState.State );

            // Slave device state
            IF (TRUE = arSlaveState[uiSlaveIdx].NotOnline) THEN
               arSlaveDevState[uiSlaveIdx] := ECAT_STATE_UNKNOWN;
            ELSIF (TRUE = NOT(arSlaveState[uiSlaveIdx].NotInStateOp)) THEN
               arSlaveDevState[uiSlaveIdx] := ECAT_STATE_OP;
            ELSE
               pDword := ADR(arSlaveState[uiSlaveIdx]);
```

```
                arSlaveDevState[uiSlaveIdx] :=
                      DWORD_TO_DINT( SHR((pDword^ AND 16#0070), 4) );
             END_IF

             // go to the next slave
             fbECATRemoteSlaveState(Execute:=FALSE);
             uiSlaveIdx := uiSlaveIdx + 1;
          END_IF

       ELSE // uiSlaveIdx
          // start again with first slave
          uiSlaveIdx := 0;
       END_IF
    END_IF // InOperation

ELSE // boEnable
    boError        := FALSE;
    boInOperation  := FALSE;
    uiSlaveIdx     := 0;
    fbECATGetConfigEthercatAddrList(Execute:=FALSE);
    fbECATRemoteSlaveState(Execute:=FALSE);
END_IF
```

Error description No specific error codes, refer to the general error codes "EtherCAT error table".

9.4.10 IL_ECATRemoteSlaveSetErrorCounterPeriod

Brief description The function block IL_ECATRemoteSlaveSetErrorCounterPeriod enables a master function to cyclically collect the error counters of all slave devices.

Interface description

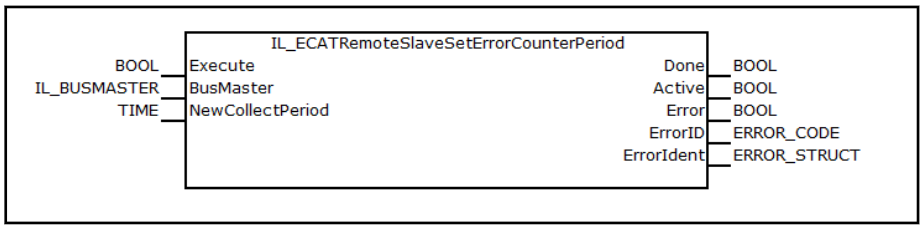


Fig. 9-15: IL_ECATRemoteSlaveSetErrorCounterPeriod function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	NewCollectPeriod	TIME	Interval to collect the error counters (disabled with T#0S), recommendation: Value > "100 ms * NumConfigSlaves".
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error

Tab. 9-30: IL_ECATRemoteSlaveSetErrorCounterPeriod interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
NewCollectPer-iod	TIME	T#0ms	N. def.	T#10s	Rising edge at “Execute”

Tab. 9-31: Min./max. and default values of the inputs IL_ECATRemoteSlaveSetErrorCounterPeriod

Functional description

The function block IL_ECATRemoteSlaveSetErrorCounterPeriod enables a master function to cyclically collect the error counters of all slave devices.

An update interval is set at the "NewCollectPeriod" input. Within that interval, the master collects the error counters from one slave after the other. It is recommended to set a "NewCollectPeriod" > "100 ms * NumConfigSlaves" to restrict the communication load of the acyclic bus accesses.

The value "NewCollectPeriod" = T#0S disables this master function.

☞ This master function to collect the error counter can be enabled/disabled by another client (e.g. MLPI).

The function block IL_ECATRemoteSlaveGetErrorCounter can be used to retrieve the collected error counters of the slaves from the EtherCAT master.

The function block IL_ECATRemoteSlaveClearErrorCounter can be used to delete the error counters in all slaves.

Implementation in ST

Example:

```
VAR
   fbECATRemoteSlaveSetErrorCounterPeriod : IL_ECATRemoteSlaveSetErrorCounterPeriod;
   boExecute : BOOL := FALSE;
   tNewCollectPeriod : TIME := T#10S;
END_VAR

fbECATRemoteSlaveSetErrorCounterPeriod(
   Execute:= boExecute,
   BusMaster:= ,
   NewCollectPeriod:= tNewCollectPeriod,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=>  );
```

Troubleshooting

No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.11 IL_ECATRemoteSlaveGetErrorCounterPeriod

Brief description

The function block IL_ECATRemoteSlaveGetErrorCounterPeriod reads the current interval of the EtherCAT master to collect the error counters of the slave devices.

Interface description

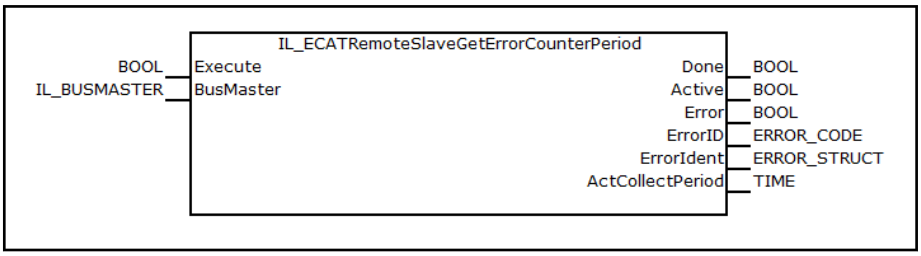


Fig. 9-16: IL_ECATRemoteSlaveGetErrorCounterPeriod function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	ActCollectPeriod	TIME	The current interval of the EtherCAT master to collect the error counters of the slave devices (disabled with T#0S)

Tab. 9-32: IL_ECATRemoteSlaveGetErrorCounterPeriod interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”

Tab. 9-33: Min./max. and default values of the inputs IL_ECATRemoteSlaveGetErrorCounterPeriod

Functional description

The function block IL_ECATRemoteSlaveGetErrorCounterPeriod reads the current interval of the EtherCAT master to collect the error counters of the slave devices.

"ActCollectPeriod" = T#0S indicates that the collecting function is disabled in the master.

The function block IL_ECATRemoteSlaveSetErrorCounterPeriod can be used to set a new interval.



As different clients (e.g. MLPI) can change this collecting interval, e.g. disable it as well, the function block IL_ECATRemoteSlaveGetErrorCounterPeriod can be used to check the current value.

Implementation in ST

Example:

```
VAR
   fbECATRemoteSlaveGetErrorCounterPeriod : IL_ECATRemoteSlaveGetErrorCounterPeriod;
   boExecute : BOOL := FALSE;
   tActCollectPeriod : TIME;
END_VAR

fbECATRemoteSlaveGetErrorCounterPeriod(
   Execute:= boExecute,
   BusMaster:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   ActCollectPeriod => tActCollectPeriod );
```

Troubleshooting

No specific error codes, refer to the general error codes "EtherCAT error table".

9.4.12 IL_ECATRemoteSlaveGetErrorCounter

Brief description

The function block IL_ECATRemoteSlaveGetErrorCounter reads out the collected error counters of a slave device from the master.

Interface description

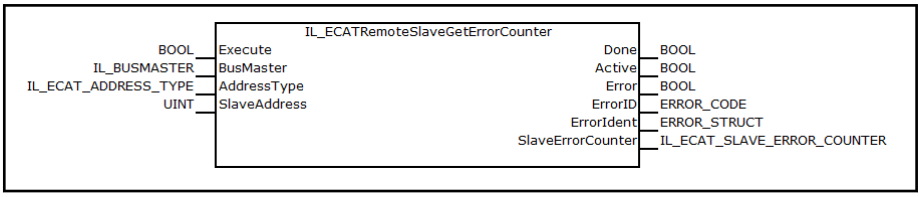


Fig. 9-17: IL_ECATRemoteSlaveGetErrorCounter function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	SlaveErrorCounter	IL_ECAT_SLAVE_ERROR_COUNTER	Error counter of the slave device

Tab. 9-34: IL_ECATRemoteSlaveGetErrorCounter interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at Execute
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”

Tab. 9-35: Min./max. and default values of the inputs IL_ECATRemoteSlaveGetErrorCounter

Functional description

The function block IL_ECATRemoteSlaveGetErrorCounter reads out the collected error counters of a slave device from the master.

The master function to collect the slave error counter has to be enabled before using the function block IL_ECATRemoteSlaveSetErrorCounterPeriod.



The function block IL_ECATRemoteSlaveGetErrorCounter does not report any errors if the master function is not enabled for collecting.

Implementation in ST *Example:*

```
VAR CONSTANT
   // max number of configured slaves, here: '128'
   uiMAX_CONFIG_SLAVES : UINT := 128;
END_VAR
VAR
   boExecute : BOOL := FALSE;
   boDone    : BOOL := FALSE;
   boError   : BOOL := FALSE;
   arEthercatAddrList  : ARRAY[0..(uiMAX_CONFIG_SLAVES-1)] OF UINT;
   arSlaveErrorCounter : ARRAY[0..(uiMAX_CONFIG_SLAVES-1)] OF IL_ECAT_SLAVE_ERROR_COUNTER;
   uiSlaveIdx : UINT := 0;
   udiNumConfigSlaves : UDINT;
   fbECATGetConfigEthercatAddrList : IL_ECATGetConfigEthercatAddrList;
   fbECATRemoteSlaveGetErrorCounter: IL_ECATRemoteSlaveGetErrorCounter;
END_VAR

// Example to read the collected slave's error counter from the Master
//   (activate with IL_ECATRemoteSlaveSetErrorCounterPeriod)

IF (boExecute = TRUE  AND  boDone = FALSE) THEN

   // get all configured EtherCAT addresses
   fbECATGetConfigEthercatAddrList(
      Execute:= TRUE,
      SizeOfEthercatAddrList:= SIZEOF(arEthercatAddrList),
      EthercatAddrListAdr:= ADR(arEthercatAddrList),
      NumConfigSlaves=> udiNumConfigSlaves );
   IF (fbECATGetConfigEthercatAddrList.Error) THEN
      boError := TRUE;
   ELSIF (fbECATGetConfigEthercatAddrList.Done = TRUE) THEN

      // read Slave ErrorCounter from one slave after the other
      IF (uiSlaveIdx < udiNumConfigSlaves) THEN

         fbECATRemoteSlaveGetErrorCounter(
            Execute:= TRUE,
            AddressType:= ECAT_ETHERCAT_ADDRESS,
            SlaveAddress:= arEthercatAddrList[uiSlaveIdx],
            SlaveErrorCounter=> arSlaveErrorCounter[uiSlaveIdx] );

         IF (TRUE = fbECATRemoteSlaveGetErrorCounter.Error) THEN
            boError := TRUE;
         ELSIF (TRUE = fbECATRemoteSlaveGetErrorCounter.Done) THEN
            // next slave
            fbECATRemoteSlaveGetErrorCounter(Execute:=FALSE);
            uiSlaveIdx := uiSlaveIdx + 1;
         END_IF

      ELSE // uiSlaveIdx
         boDone := TRUE; // Finished
      END_IF
   END_IF

ELSIF (boExecute = FALSE) THEN
   boDone := FALSE;
   boError:= FALSE;
   uiSlaveIdx := 0;
   fbECATGetConfigEthercatAddrList(Execute:=FALSE);
   fbECATRemoteSlaveGetErrorCounter(Execute:=FALSE);
END_IF
```

Error description No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.13 IL_ECATRemoteSlaveClearErrorCounter

Brief description The function block IL_ECATRemoteSlaveClearErrorCounter deletes all error counters in all slave devices.

Interface description

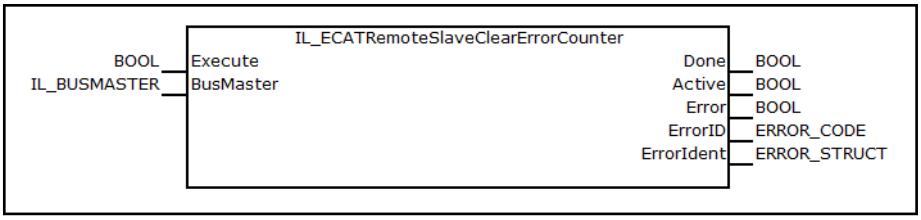


Fig. 9-18: IL_ECATRemoteSlaveClearErrorCounter function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error

Tab. 9-36: IL_ECATRemoteSlaveClearErrorCounter interface

Name	Data type	Min	Max	Default	Effective
Enable	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”

Tab. 9-37: Min./max. and default values of the inputs IL_ECATRemoteSlaveClearErrorCounter

Functional description

The function block IL_ECATRemoteSlaveClearErrorCounter deletes all error counters in all slave devices.

Implementation in ST

Example:

```
VAR
   fbECATRemoteSlaveClearErrorCounter: IL_ECATRemoteSlaveClearErrorCounter;
   boExecute : BOOL := FALSE;
END_VAR

fbECATRemoteSlaveClearErrorCounter(
   Execute:= boExecute,
   BusMaster:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=>  );
```

Error description

No specific error codes, refer to the general error codes “EtherCAT error table“.

9.4.14 IL_ECATCoeReadDiagHistory

Brief description

The function block IL_ECATCoeReadDiagHistory can be used to acyclically retrieve the CoE "Diagnosis History" Object (index 16#10F3) from an EtherCAT slave device.

Interface description

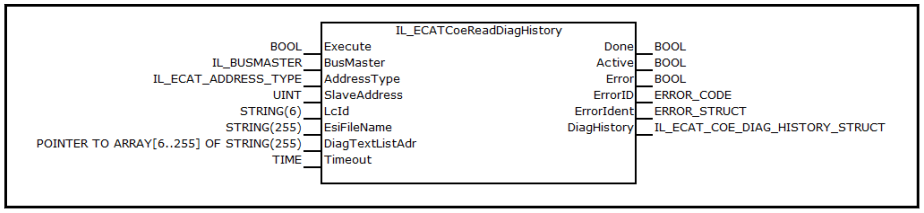


Fig. 9-19: IL_ECATCoeReadDiagHistory function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
	LcId	STRING(6)	[Optional] "Local Id" (language code identifier): Language ID, see ESI file, which languages are supported for diagnostic texts (e.g. "1033" = EN-US, "1031" = DE-DE)
	EsiFileName	STRING(255)	[Optional] file name of the ESI file incl. path (on the flash card of the control)
	DiagTextListAdr	POINTER TO ARRAY[6..255] OF STRING(255)	[Optional] Diagnostic text list on the diagnostic messages. Attention: The data type has to fully match, as the array [index] interrelates with the "DiagMessages" index.
	Timeout	TIME	Timeout monitoring for the internally used acyclic services.
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	DiagHistory	IL_ECAT_COE_DIAG_HISTORY_STRUCT	"Diagnosis History" object (index 16#10F3) of the slave device

Tab. 9-38: IL_ECATCoeReadDiagHistory interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”

Name	Data type	Min	Max	Default	Effective
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at Execute
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”
LcId	STRING(6)	-	-	‚1033‘	Rising edge at “Execute”
EsiFileName	STRING(255)	-	-	‚‘	Rising edge at “Execute”
DiagTextListAdr	POINTER TO ARRAY[6..255] OF STRING(255)	N. def.	N. def.	0	Continuous
Timeout	TIME	T#1ms	N. def.	T#3s	Rising edge at “Execute”

Tab. 9-39: Min./max. and default values of the inputs IL_ECATCoeReadDiagHistory

Functional description

The function block IL_ECATCoeReadDiagHistory can be used to acyclically retrieve the CoE "Diagnosis History" object (index 16#10F3) from an EtherCAT slave device.

It is required that the slave device supports the mailbox protocol "CAN application protocol over EtherCAT" (CoE). Mailbox communication is only possible in the EtherCAT states PreOP, SafeOP and OP.

☞ The "Diagnosis History" object (Index 16#10F3) is optional and it is not supported by each CoE slave. The object also provides optional functions. For the functions supported by the slave, refer to the product documentation or the device description (ESI) of the slave device.

General:

- There are different types of diagnostic messages: Info, warning, error
- Up to 250 diagnostic messages can be saved in the slave device (ring buffer)
- A diagnostic message contains a "TextId" as reference for a detailed error description in the device description (ESI)
- The function block does not represent diagnostic messages (SI6-255) as raw data, but in an extended data structure to allow an easy evaluation
- Two modes are distinguished:
  - "Overwrite Mode"
  - "Acknowledge mode"

  The active mode is displayed at the "flags" (SI5, bit 0.4). Switching the mode (write access) is not supported by all slaves
- Normally, the "NewDiagMessages" bit (SI4) is part of the cyclic communication (PDO configuration) of the slaves to display if a new diagnostic message is present

"**Overwrite mode**":

- If the message buffer is full, a new message overwrites the oldest message in the buffer. Unconfirmed messages are also overwritten
- Confirming messages:

By writing "NewestAckMessage" (SI3), all older messages are confirmed up to the value (SubIndex) they are written, irrespective of whether these were read before

- Deleting messages:

  By writing "NewestAckMessage" (SI3) with "0", all messages are deleted also if these are not confirmed or unread (optional functionality as in the slave)

- If a SubIndex value is written not containing any message, the following CoE abort codes can result:
  - 0x06090030
  - 0x06090031
  - 0x06090032

"Acknowledge mode":

- Unconfirmed messages are not overwritten. E.g. if the message buffer is full, the latest messages are not saved and thus lost
- Confirming messages:

  By writing "NewestAckMessage" (SI3), all older messages are confirmed up to the value (SubIndex) they are written, irrespective of whether these were read before

- Deleting messages:

  By writing "NewestAckMessage" (SI3) with "0", all messages confirmed before are deleted (optional functionality in the slave)

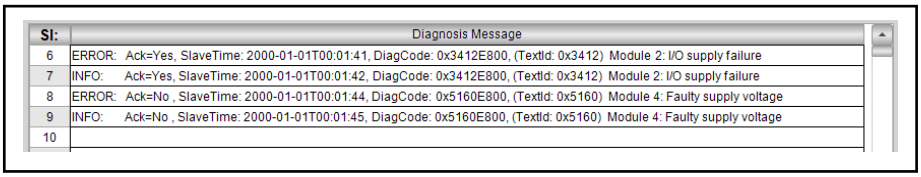
- If a SubIndex value is written not containing any message, the following CoE abort codes can result:
  - 0x06090030
  - 0x06090031
  - 0x06090032

Additional information:

- To write an SDO SubIndex, the function block IL_ECATCoeWriteSdo can be used with the following parameters:
  - Index = 16#10F3 ("Diagnosis History" object)
  - SubIndex = 3 (SI3)
- When writing "flags" (SI5) to set the operation modes of the diagnostic object, note that it might be possible that some control bits of the "flags" cannot be changed by the slave device. In case of these control bits, the write value has to correspond to the actual value. Otherwise, the following CoE abort codes can result:
  - 0x06090030 and others
- ESI file:
  - The "EsiFileName" input can be used optionally. The ESI file has to be stored manually on the flash card of the control. The file name has to be specified including the path.

    For example on the USER partition in a subfolder ESI: strEsiFileName: STRING := '/USER/ESI/S20-EC-BK.xml'
  - The function block does not check whether the specified ESI matches the addressed slave device. The first "DiagMessages" entry in the ESI is used

- The ESI has to be in UTF-8. If it is in another format, convert that format if required
- If the specified language ID (input "LcId") is not found in ESI, the LcId value “1033” (EN-US) is tried. If it is also not found, an error results
- There are the following restrictions for the function block for the generic parameter list of a diagnostic message referenced in the "MessageText" from the ESI:
  - (a) Up to 12 parameters are supported
  - (b) The parameters have to be referenced in ascending order in "MessageText"



SI:	Diagnosis Message
6	ERROR: Ack=Yes, SlaveTime: 2000-01-01T00:01:41, DiagCode: 0x3412E800, (TextId: 0x3412) Module 2: I/O supply failure
7	INFO: Ack=Yes, SlaveTime: 2000-01-01T00:01:42, DiagCode: 0x3412E800, (TextId: 0x3412) Module 2: I/O supply failure
8	ERROR: Ack=No , SlaveTime: 2000-01-01T00:01:44, DiagCode: 0x5160E800, (TextId: 0x5160) Module 4: Faulty supply voltage
9	INFO: Ack=No , SlaveTime: 2000-01-01T00:01:45, DiagCode: 0x5160E800, (TextId: 0x5160) Module 4: Faulty supply voltage
10	

Fig. 9-20: Exemplary screenshot for the representation of the diagnostic message (input "DiagTextListAdr")

Implementation in ST *Example:*

```
VAR
   fbECATCoeReadDiagHistory: IL_ECATCoeReadDiagHistory;
   boExecute : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   // ESI example: you have to store manually the corresponding ESI file
   //     on the compact flash of the control,
   //     here: USER-partition with subfolder 'ESI'
   strEsiFileName : STRING := '/USER/ESI/S20-EC-BK.xml';
   // see ESI which LcId is supported, e.g. '1033' = EN-US; '1031' = DE-DE
   strLcId : STRING(6) := '1033';
   arDiagTextList : ARRAY[6..255] OF STRING(255);
END_VAR

// Example to read CoE 'Diagnosis History' (16#10F3)
// Note: Slave must support CoE and the 'Diagnosis History' (16#10F3) object
fbECATCoeReadDiagHistory(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   LcId:= strLcId,
   // assign only if ESI corresponds to your device
   EsiFileName:= , // strEsiFileName,
   DiagTextListAdr := ADR(arDiagTextList),
   Timeout:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   DiagHistory=> );
```

Error description Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#F0EC8000	16#00000004	Input "Timeout" invalid: Value "T#0ms" not permitted
ACCESS_ERROR	16#F0EC8020	Internal	Error when opening file. Check input "EsiFileName" (file name including path)
ACCESS_ERROR	16#F0EC8021	Internal	Error when reading file (internal)

ErrorID	Additional1	Additional2	Description
ACCESS_ERROR	16#F0EC8022	Internal	Error when setting file position (internal)
ACCESS_ERROR	16#F0EC8023	Internal	a.) File tag <DiagMessages> not found, Check ESI file whether <DiagMessages> are supported. b.) Only a UTF-8 coding is supported. If the coding of the ESI file is different, try to convert this file into UTF-8.

Also refer to the general error codes “EtherCAT error table“.

9.5 State machine

9.5.1 IL_ECATMasterGetBusState

Brief description The function block IL_ECATMasterGetBusState returns the current bus state of the EtherCAT master.

Interface description

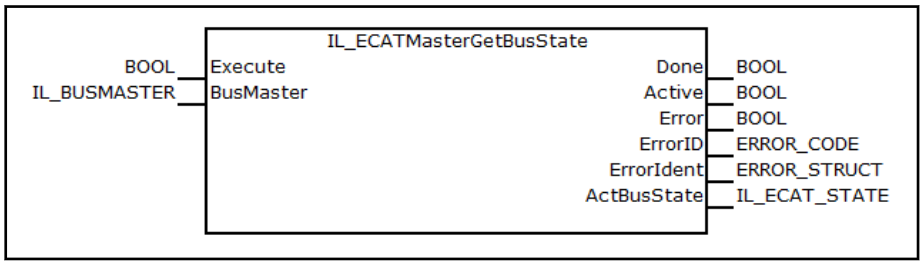


Fig. 9-21: IL_ECATMasterGetBusState function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	ActBusState	IL_ECAT_STATE	Current bus state of the EtherCAT master

Tab. 9-40: IL_ECATMasterGetBusState interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”

Tab. 9-41: Min./max. and default values of the inputs IL_ECATMasterGetBusState

Functional description The function block IL_ECATMasterGetBusState returns the current bus state of the EtherCAT master.

Implementation in ST *Example:*

```
VAR
   fbECATMasterGetBusState: IL_ECATMasterGetBusState;
   boExecute : BOOL := FALSE;
   eActBusState : IL_ECAT_STATE;
END_VAR

fbECATMasterGetBusState(
   Execute:= boExecute,
   BusMaster:= ,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   ActBusState=> eActBusState );
```

Error description No specific error codes, refer to the general error codes “EtherCAT error table“.

9.5.2 IL_ECATMasterSetBusState

Brief description The function block IL_ECATMasterSetBusState can be used to request an EtherCAT bus state at the EtherCAT master.

Interface description

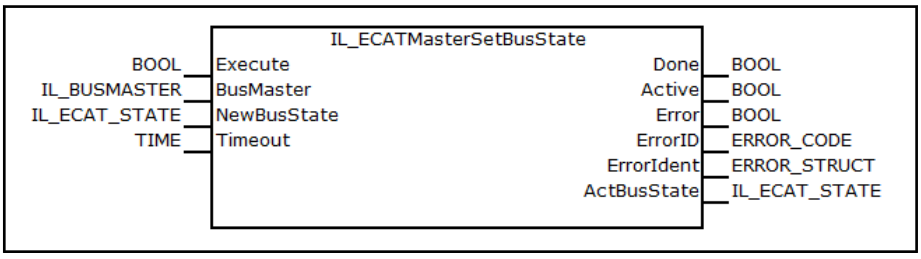


Fig. 9-22: IL_ECATMasterSetBusState function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	NewBusState	IL_ECAT_STATE	EtherCAT bus state requested
	Timeout	TIME	Timeout monitoring (disabled with T#0s)
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	ActBusState	IL_ECAT_STATE	Current bus state of the EtherCAT master

Tab. 9-42: IL_ECATMasterSetBusState interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”

Name	Data type	Min	Max	Default	Effective
NewBusState	IL_ECAT_STATE	ECAT_STATE_INIT	ECAT_STATE_OP	ECAT_STATE_UNKNOWN	Rising edge at “Execute”
Timeout	TIME	T#0s	N. def.	T#0s	Rising edge at “Execute”

Tab. 9-43: Min./max. and default values of the inputs IL_ECATMasterSetBusState

Functional description

The function block IL_ECATMasterSetBusState can be used to request an EtherCAT bus state at the EtherCAT master.

The EtherCAT master then tries to switch the bus state and the state of all slaves to the requested state.

The following EtherCAT states can be requested:

- Init
- PreOP
- SafeOP
- OP

If the requested bus state is not reached within the timeout monitoring ("Timeout" > T#0s), the function block reports an error, but the internal command execution cannot be stopped!

If the master is in a defined state (e.g. OP), but some slaves are not in this state, the IL_ECATMasterSetBusState can also be called with this target state. The master then tries to switch all slaves to the requested state.

Note that the output "ActBusState" only displays the bus state. To determine whether all slaves are in the master state, the diagnostic function block IL_ECATMasterState can be used.

Implementation in ST

Example:

```
VAR
   fbECATMasterSetBusState: IL_ECATMasterSetBusState;
   boExecute: BOOL := FALSE;
   eNewBusState: IL_ECAT_STATE := ECAT_STATE_OP;
   tTimeout: TIME := T#30S;
END_VAR

// Example to request a new bus state
fbECATMasterSetBusState(
   Execute:= boExecute,
   BusMaster:= ,
   NewBusState:= eNewBusState,
   Timeout:= tTimeout,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   ActBusState=> );
```

Error description

Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
ACCESS_ERROR	16#F0ED000B	Internal	Input "NewBusState" invalid: Invalid parameter value

Also refer to the general error codes “Error codes error table“.

9.5.3 IL_ECATRemoteSlaveGetDevState

Brief description

The function block IL_ECATRemoteSlaveGetDevState returns the current EtherCAT state of a slave.

Interface description

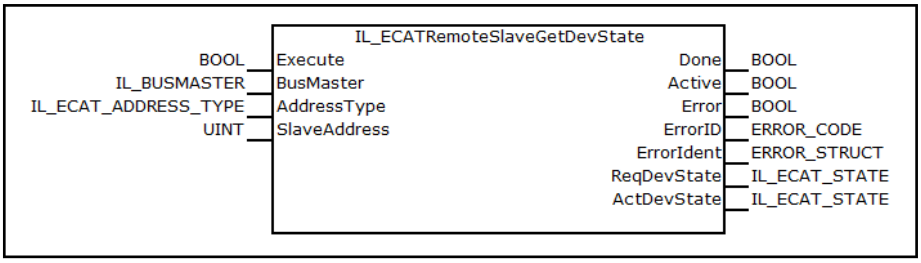


Fig. 9-23: IL_ECATRemoteSlaveGetDevState function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	ReqDevState	IL_ECAT_STATE	Requested EtherCAT state of the slave device
	ActDevState	IL_ECAT_STATE	Current EtherCAT slave of the slave device

Tab. 9-44: IL_ECATRemoteSlaveGetDevState interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
AddressType	IL_ECAT_ADDRESS_TYPE	ECAT_AUTOINC_ADDRESS	ECAT_ETHERCAT_ADDRESS	ECAT_ETHERCAT_ADDRESS	Rising edge at Execute
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”

Tab. 9-45: Min./max. and default values of the inputs IL_ECATRemoteSlaveGetDevState

Functional description

The function block IL_ECATRemoteSlaveGetDevState returns the current EtherCAT state of a slave.

Implementation in ST

Example:

```
VAR
   fbECATRemoteSlaveGetDevState: IL_ECATRemoteSlaveGetDevState;
```

```
   boExecute : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   eActDevState : IL_ECAT_STATE;
   eReqDevState : IL_ECAT_STATE;
END_VAR

fbECATRemoteSlaveGetDevState(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
   ReqDevState=> eReqDevState,
   ActDevState=> eActDevState );
```

Error description No specific error codes, refer to the general error codes “EtherCAT error table“.

9.5.4 IL_ECATRemoteSlaveSetDevState

Brief description The function block IL_ECATRemoteSlaveSetDevState can be used to request an EtherCAT state at a slave device.

Interface description

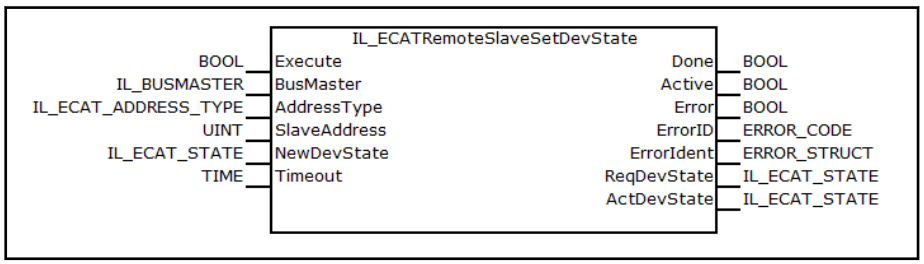


Fig. 9-24: IL_ECATRemoteSlaveSetDevState function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Instance of the EtherCAT master in the order of the configuration tree
	AddressType	IL_ECAT_ADDRESS_TYPE	Selection of the slave addressing (EtherCAT address or AutoInc address)
	SlaveAddress	UINT	EtherCAT address or the current AutoInc address at the bus (depends on the input "AddressType")
	NewDevState	IL_ECAT_STATE	EtherCAT state requested at the slave device
	Timeout	TIME	Timeout monitoring (disabled with T#0s)
VAR_OUTPUT	Done	BOOL	Processing completed without error, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error, output data invalid
	ErrorID	ERROR_CODE	Standardized rough classification of the error
	ErrorIdent	ERROR_STRUCT	Function block-specific detailed information on the error
	ReqDevState	IL_ECAT_STATE	Requested EtherCAT state of the slave device
	ActDevState	IL_ECAT_STATE	Current EtherCAT slave of the slave device

Tab. 9-46: IL_ECATRemoteSlaveSetDevState interface

Name	Data type	Min	Max	Default	Effective
Execute	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
AddressType	IL_ECAT_AD-DRESS_TYPE	ECAT_AUTO-INC_ADDRESS	ECAT_ETHER-CAT_ADDRESS	ECAT_ETHER-CAT_AD-DRESS	Rising edge at Execute
SlaveAddress	UINT	0	65535	0	Rising edge at “Execute”
NewDevState	IL_ECAT_STA TE	ECAT_STATE_IN IT	ECAT_STATE_ OP	ECAT_STATE_ UNKNOWN	Rising edge at “Execute”
Timeout	TIME	T#0s	N. def.	T#0s	Rising edge at “Execute”

Tab. 9-47: Min./max. and default values of the inputs IL_ECATRemoteSlave-SetDevState

Functional description

The function block IL_ECATRemoteSlaveSetDevState can be used to request an EtherCAT state at a slave device.

The following EtherCAT states can be requested:

- Init
- PreOP
- Bootstrap
- SafeOP
- OP

There are the following boundary conditions:

- The requested state in a slave device may not be higher than the one in the EtherCAT master bus state (e.g. if the master is in PreOP, a slave device cannot be switched to SafeOP or OP)
- The EtherCAT slave state "Bootstrap" is optional in the slave devices. The state "Bootstrap" can only be reached from the EtherCAT slave state "Init"
- If the requested bus state is not reached within the timeout monitoring ("Timeout" > T#0s), the function block reports an error, but the internal command execution cannot be stopped!

Implementation in ST

Example:

```
VAR
   fbECATRemoteSlaveSetDevState : IL_ECATRemoteSlaveSetDevState;
   boExecute : BOOL := FALSE;
   uiEthercatAddr : UINT := 1001;
   eNewDevState : IL_ECAT_STATE := ECAT_STATE_PREOP;
   tTimeout: TIME := T#20S;
END_VAR

// Example to request a new EtherCAT state at a slave device
fbECATRemoteSlaveSetDevState(
   Execute:= boExecute,
   BusMaster:= ,
   AddressType:= ,
   SlaveAddress:= uiEthercatAddr,
   NewDevState:= eNewDevState,
   Timeout:= tTimeout,
   Done=> ,
   Active=> ,
   Error=> ,
   ErrorID=> ,
   ErrorIdent=> ,
```

```
ReqDevState=> ,
ActDevState=> );
```

Error description Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
ACCESS_ERROR	16#F0ED000B	Internal	Input "NewDevState" invalid: Invalid parameter value
ACCESS_ERROR	16#F0ED000E	Internal	Input "NewDevState" invalid: Invalid change in state

Also refer to the general error codes “EtherCAT error table“.

9.6 Auxiliary functions (utilities)

9.6.1 IL_ECATSoeIdnCoding

Brief description The function IL_ECATSoeIdnCoding summarizes the following individual elements in one parameter identification number: Parameter type, parameter block and data block number.

Interface description

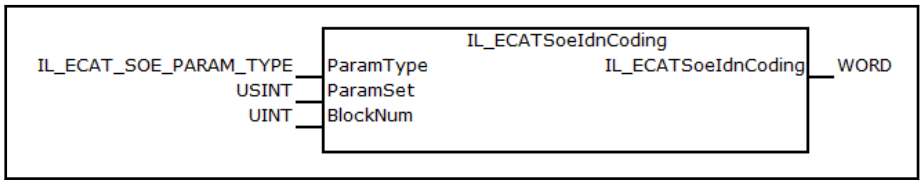


Fig. 9-25: IL_ECATSoeIdnCoding function

I/O type	Name	Data type	Description
VAR_INPUT	ParamType	IL_ECAT_SOE_PARAM_TYPE	Parameter type (S- or P-parameter)
	ParamSet	USINT	Parameter block (0..7)
	BlockNum	UINT	Data block number
RETURN VALUE	IL_ECATSoeIdnCoding	WORD	Parameter identification number

Functional description The function IL_ECATSoeIdnCoding summarizes the following individual elements in one parameter identification number: Parameter type, parameter block and data block number.

Implementation in ST *Example:*

```
fbSoeRead.Idn := IL_ECATSoeIdnCoding(SOE_S_PARAM, 0, 1504); // ^= S-0-1504
```

Error description The function IL_ECATSoeAttributeDecoding is not provided with any troubleshooting.

9.6.2 IL_ECATSoeAttributeDecoding

Brief description The function IL_ECATSoeAttributeDecoding decodes the SoE parameter "Attribute".

Interface description

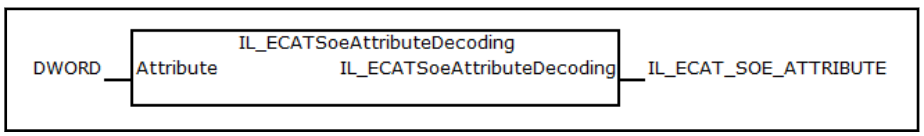


Fig. 9-26: IL_ECATSoeAttributeDecoding function

I/O type	Name	Data type	Description
VAR_INPUT	Attributes	DWORD	SoE parameter "Attribute" value
RETURN VALUE	IL_ECATSoeAttributeDecoding	IL_ECAT_SOE_ATTRIBUTE	Parameter attribute as structure type

Functional description The function IL_ECATSoeAttributeDecoding decodes the SoE parameter "Attribute".

Implementation in ST *Example:*

```
VAR
   dwAttribute : DWORD; // read value
   rAttribute  : IL_ECAT_SOE_ATTRIBUTE;
END_VAR

rAttribute := IL_ECATSoeAttributeDecoding( dwAttribute );
```

Error description The function IL_ECATSoeAttributeDecoding is not provided with any troubleshooting.

9.6.3 IL_ECATMasterStateDecoding

Brief description The function IL_ECATMasterStateDecoding decodes the master "State" of the function block IL_ECATMasterState.

Interface description

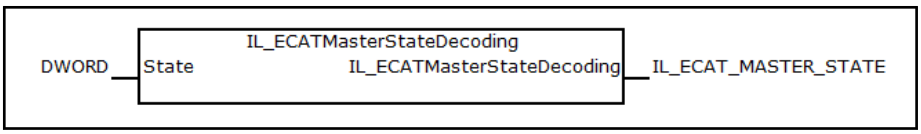


Fig. 9-27: IL_ECATMasterStateDecoding function

I/O type	Name	Data type	Description
VAR_INPUT	State	DWORD	Master "State" of the function block IL_ECATMasterState
RETURN VALUE	IL_ECATMasterStateDecoding	IL_ECAT_MASTER_STATE	Bit structure to comfortably access the master "State"

Functional description The function IL_ECATMasterStateDecoding decodes the master "State" of the function block IL_ECATMasterState.

Implementation in ST *Example:*

```
VAR
   rMasterState : IL_ECAT_MASTER_STATE;
END_VAR

rMasterState := IL_ECATMasterStateDecoding( fbECATMasterState.State );
```

Error description The function IL_ECATMasterStateDecoding is not provided with any troubleshooting.

9.6.4 IL_ECATSlaveStateDecoding

Brief description The function IL_ECATSlaveStateDecoding decodes the slave "State" of the function block IL_ECATRemoteSlaveState.

Interface description

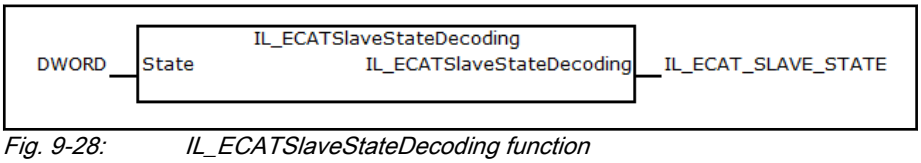


Fig. 9-28: IL_ECATSlaveStateDecoding function

I/O type	Name	Data type	Description
VAR_INPUT	State	DWORD	Slave "State" of the function block IL_ECATRemoteSlaveState
RETURN VALUE	IL_ECATSlaveState-Decoding	IL_ECAT_SLAVE_STATE	Bit structure to comfortably access the slave "State"

Functional description The function IL_ECATSlaveStateDecoding decodes the slave "State" of the function block IL_ECATRemoteSlaveState.

Implementation in ST *Example:*

```
VAR
   rSlaveState : IL_ECAT_SLAVE_STATE;
END_VAR

rSlaveState := IL_ECATSlaveStateDecoding( fbECATRemoteSlaveState.State );
```

Error description The function IL_ECATSlaveStateDecoding is not provided with any trouble-shooting.

9.7 Data types (DataTypes)

9.7.1 IL_BUSMASTER

Brief description The enumeration type IL_BUSMASTER of the RIL_FieldbusTypes.library is used to select the respective EtherCAT master.

Only one EtherCAT master instance is currently supported.

Element	Value	Description
IL_BUSMASTER_0	0	First EtherCAT master
IL_BUSMASTER_1	1	Not supported
IL_BUSMASTER_2	2	Not supported
IL_BUSMASTER_3	3	Not supported
IL_BUSMASTER_4	4	Not supported

Tab. 9-48: Elements of the IL_BUSMASTER enumeration type

9.7.2 IL_ECAT_ADDRESS_TYPE

Brief description The enumeration type IL_ECAT_ADDRESS_TYPE describes the EtherCAT addressing type.

Name	Value	Description
ECAT_AUTOINC_ADDRESS	0	Auto-increment address
ECAT_ETHERCAT_ADDRESS	1	EtherCAT address ("fixed physical address")

Tab. 9-49: Elements of the IL_ECAT_ADDRESS_TYPE enumeration type

9.7.3 IL_ECAT_BASE_DATA_TYPE

Brief description The enumeration type IL_ECAT_BASE_DATA_TYPE describes the data types acc. of the ETG coding.

Name	Value	Description
ECAT_DATATYPE_BOOLEAN	16#0001	BOOL
ECAT_DATATYPE_INTEGER8	16#0002	Short Integer
ECAT_DATATYPE_INTEGER16	16#0003	Integer
ECAT_DATATYPE_INTEGER32	16#0004	Double Integer
ECAT_DATATYPE_UNSIGNED8	16#0005	Unsigned Short Integer
ECAT_DATATYPE_UNSIGNED16	16#0006	Unsigned Integer
ECAT_DATATYPE_UNSIGNED32	16#0007	Unsigned Double Integer
ECAT_DATATYPE_REAL32	16#0008	Floating Point
ECAT_DATATYPE_VISIBLE_STRING	16#0009	STRING(n), 1 byte per character
ECAT_DATATYPE_OCTET_STRING	16#000A	ARRAY(0..n) OF BYTE, 1 byte per character (+1)
ECAT_DATATYPE_UNICODE_STRING	16#000B	ARRAY(0..n) OF WORD, 2 bytes per character (+1)
ECAT_DATATYPE_TIME_OF_DAY	16#000C	-
ECAT_DATATYPE_TIME_DIFFERENCE	16#000D	-
ECAT_DATATYPE_DOMAIN	16#000F	-
ECAT_DATATYPE_INTEGER24	16#0010	-
ECAT_DATATYPE_REAL64	16#0011	Long Float
ECAT_DATATYPE_INTEGER40	16#0012	-
ECAT_DATATYPE_INTEGER48	16#0013	-
ECAT_DATATYPE_INTEGER56	16#0014	-
ECAT_DATATYPE_INTEGER64	16#0015	Long Integer
ECAT_DATATYPE_UNSIGNED24	16#0016	-
ECAT_DATATYPE_UNSIGNED40	16#0018	-
ECAT_DATATYPE_UNSIGNED48	16#0019	-
ECAT_DATATYPE_UNSIGNED56	16#001A	-
ECAT_DATATYPE_UNSIGNED64	16#001B	Unsigned Long Integer
ECAT_DATATYPE_GUID	16#001D	Globally unique identifier, 128 bit
ECAT_DATATYPE_BYTE	16#001E	1 Octet
ECAT_DATATYPE_WORD	16#001F	2 Octet
ECAT_DATATYPE_DWORD	16#0020	4 Octet
ECAT_DATATYPE_PDO_MAPPING	16#0021	-
ECAT_DATATYPE_IDENTITY	16#0023	-
ECAT_DATATYPE_COMMAND_PAR	16#0025	-
ECAT_DATATYPE_PDO_PARAMETER	16#0027	-
ECAT_DATATYPE_ENUM	16#0028	-
ECAT_DATATYPE_SYNC_PAR	16#0029	-
ECAT_DATATYPE_RECORD	16#002A	-

Name	Value	Description
ECAT_DATATYPE_BACKUP_PARAMETER	16#002B	-
ECAT_DATATYPE_MOD_DEV_PROFILE	16#002C	Modular device profile
ECAT_DATATYPE_BITARR8	16#002D	8 individual bits
ECAT_DATATYPE_BITARR16	16#002E	16 individual bits
ECAT_DATATYPE_BITARR32	16#002F	32 individual bits
ECAT_DATATYPE_BIT1	16#0030	-
ECAT_DATATYPE_BIT2	16#0031	-
ECAT_DATATYPE_BIT3	16#0032	-
ECAT_DATATYPE_BIT4	16#0033	-
ECAT_DATATYPE_BIT5	16#0034	-
ECAT_DATATYPE_BIT6	16#0035	-
ECAT_DATATYPE_BIT7	16#0036	-
ECAT_DATATYPE_BIT8	16#0037	-

Tab. 9-50: Elements of the IL_ECAT_BASE_DATA_TYPE enumeration type

9.7.4 IL_ECAT_COE_DIAG_HISTORY_FLAGS

Brief description The structure IL_ECAT_COE_DIAG_HISTORY_FLAGS describes the settings on the operating principle of the CoE "Diagnosis History" object.

Structural element	Data type	Description
EnableEmergencySending	BIT	Bit 0.0: Sending acc. to ETG.1000-6 [R(/W[1])]
DisableInfoMessages	BIT	Bit 0.1: Disabling "Info" messages [R/W]
DisableWarningMessages	BIT	Bit 0.2: Disabling "warning" messages [R/W]
DisableErrorMessages	BIT	Bit 0.3: Disabling "error" messages [R(/W[3]))]
ModeSelection	BIT	Bit 0.4: Mode: 0 = "Overwrite mode", 1 = "Acknowledge mode" [R(/W[1])]
OverwriteDiscardInfo	BIT	Bit 0.5: Buffer overflow, messages were overwritten ("Overwrite Mode") or discarded ("Acknowledge Mode") [R]
Res06 … Res17	BIT … BIT	Bit 0.6: Reserved … Bit 1.7: Reserved

Tab. 9-51: Elements of the IL_ECAT_COE_DIAG_HISTORY_FLAGS structure

9.7.5 IL_ECAT_COE_DIAG_HISTORY_STRUCT

Brief description The structure IL_ECAT_COE_DIAG_HISTORY_STRUCT describes the data structure of the CoE "Diagnosis History" object (index 16#10F3). The diagnostic messages (SI6: "DiagMessages") are provided with an extended data structure and do not correspond to the raw data of the object (refer to chapter 9.4.14 "IL_ECATCoeReadDiagHistory" on page 290).

3) Write access is optional/device-specific.

Structural element	Data type	Description
NumOfEntries	BYTE	S10: Number of entries (max. SubIndex)
MaxDiagMessages	BYTE	SI1: Max. number of diagnostic messages (from SI6)
NewestMessage	BYTE	SI2: SubIndex of the latest diagnostic message [6..255] (default = 0)
NewestAckMessage	BYTE	SI3: [R/W] Meaning depends on the mode ("Overwrite Mode" or "Acknowledge Mode"), see in the following
NewDiagMessages	BYTE	SI4: Value 1 = New diagnostic messages available: "Overwrite mode": Messages not yet read "Acknowledge mode": Messages not yet confirmed
Flags	IL_ECAT_COE_DIAG_HISTORY_FLAGS	SI5: [R(/W)] Settings on the operating principle of the objects
DiagMessages	ARRAY[6..255] OF IL_ECAT_COE_DIAG_MSG_EXT	SI6 (SI7..255 optional): Diagnostic messages, max. 250 entries

Tab. 9-52: Elements of the IL_ECAT_COE_DIAG_HISTORY_STRUCT structure

NewestAckMessage (SI3) The meaning depends on the mode:

Read/Write SI3	Writing “Overwrite Mode” (SI5:Bit4 = 0)
Read = 0	If the message buffer is overwritten, the slave SI3 sets = 0
Write = 0	(Optional) slave deletes all messages (SI2, SI3, SI4 and SI5:Bit5 are reset)
Write = 1..5	The slave reports an error (e.g. SDO abort code 0x06090030 or 0x06090031)
Write = 6..[SI0]	All messages are acknowledged up to the written value
Write > [SI0]..255	The slave reports an error (e.g. SDO abort code 0x06090030 or 0x06090031)

Tab. 9-53: Overwrite Mode

Read/Write SI3	Writing Acknowledge Mode (SI5:Bit4 = 1)
Read = 0	No messages have yet been acknowledged
Read <> 0	SubIndex of the latest acknowledged message (6..255)
Write = 0	(Optional) all acknowledged messages are deleted
Write = 1..5	The slave reports an error (e.g. SDO abort code 0x06090030 or 0x06090031)
Write = 6..[SI0]	All messages are acknowledged up to the written value
Write > [SI0]..255	The slave reports an error (e.g. SDO abort code 0x06090030 or 0x06090031)

Tab. 9-54: Acknowledge mode

9.7.6 IL_ECAT_COE_DIAG_MSG_EXT

Brief description The structure IL_ECAT_COE_DIAG_MSG_EXT describes an extended data structure for a diagnostic message of the CoE "Diagnosis History" object (refer to chapter 9.4.14 "IL_ECATCoeReadDiagHistory" on page 290).

Structural element	Data type	Description
DiagCode_LWord	WORD	LWord (bits 0-15) | HWord (bits 16-31) 0x0000 - 0xDFFF → Not used 0xE000 - 0xE7FF → Vendor-specific 0xE800 → Emergency ErrorCode acc. to DS301 or DS4xxx 0xE801 - 0xEDFF → Reserved 0xEE00 - 0xEFFF → Profile-specific 0xF000 - 0xFFFF → Not used
DiagCode_HWord	WORD	
DiagType	IL_ECAT_COE_DIAG_TYPE	Diagnostic type ("flags" bits 0-3)
Acknowledged	BOOL	TRUE = Message is confirmed
TimeStampNs	ULINT	Time stamp in ns; DC or local time stamp if DC 64 bit is not supported (0 = no time stamp)
IsLocalTimeStamp	BOOL	Time stamp is local from slave device ("flags" bit 4)
TextId	WORD	Text ID as reference to the diagnostic text in the ESI file (0 = no ID)
NumOfParams	BYTE	Number of parameters in the diagnostic message ("flags" bits 8-15)
Data	ARRAY[0..127] OF BYTE	[Optional] data of parameter 1 and more parameters
MessageText	STRING(127)	[Optional] "MessageText" of the ESI (if an ESI file is specified)

Tab. 9-55: Elements of the IL_ECAT_COE_DIAG_MSG_EXT structure

9.7.7 IL_ECAT_COE_DIAG_MSG_RAW

Brief description The structure IL_ECAT_COE_DIAG_MSG_RAW describes raw data of a diagnostic message of the CoE "Diagnosis History" object.

Structural element	Data type	Description
DiagCode_LWord	WORD	LWord (bits 0-15) | HWord (bits 16-31) 0x0000 - 0xDFFF → Not used 0xE000 - 0xE7FF → Vendor-specific 0xE800 → Emergency ErrorCode acc. to DS301 or DS4xxx 0xE801 - 0xEDFF → Reserved 0xEE00 - 0xEFFF → Profile-specific 0xF000 - 0xFFFF → Not used
DiagCode_HWord	WORD	
Flags_LByte	BYTE	Bit 0-3: Diagnostic type (0=Info, 1=Warning, 2=Error) Bit 4: Time stamp is local from slave device. Bit 5-7: Reserved.
Flags_HByte	BYTE	Number of parameters in the diagnostic message
TextId	WORD	Text ID as reference to the diagnostic text in the ESI file (0 = no ID)
TimeStamp	ULINT	Time stamp in ns: DC or local time stamp if DC 64 bit is not supported (0 = no time stamp)
Data	ARRAY[0..127] OF BYTE	(Optional) data of parameter 1 and more parameters

Tab. 9-56: Elements of the IL_ECAT_COE_DIAG_MSG_RAW structure

9.7.8 IL_ECAT_COE_DIAG_TYPE

Brief description The enumeration type IL_ECAT_COE_DIAG_TYPE describes the diagnostic type of a CoE diagnostic message.

Name	Value	Description
COE_DIAG_INFO	0	Diagnostics "Info" message
COE_DIAG_WARNING	1	Diagnostics "Warning" message
COE_DIAG_ERROR	2	Diagnostics "Error" message

Tab. 9-57: Elements of the IL_ECAT_COE_DIAG_TYPE enumeration type

9.7.9 IL_ECAT_DL_STATUS

Brief description The structure IL_ECAT_DL_STATUS describes the data link status of an EtherCAT slaves (ESC register 0x0110:0x0111).

Structural element	Data type	Description
PdiOperationalEepromLoaded	BIT	Bit 0.0: PDI functional, EEPROM loaded correctly
PdiWatchdogStatus	BIT	Bit 0.1: Watchdog loaded/running
EnhancedLinkDetection	BIT	Bit 0.2: Enhanced link detection for at least one port
Reserved03	BIT	Bit 0.3: Reserved
LinkPort0	BIT	Bit 0.4: Link available at port 0
LinkPort1	BIT	Bit 0.5: Link available at port 1
LinkPort2	BIT	Bit 0.6: Link available at port 2
LinkPort3	BIT	Bit 0.7: Link available at port 3
LoopClosedPort0	BIT	Bit 1.0: Port 0 is closed
CommunicationPort0	BIT	Bit 1.1: Communication at port 0
LoopClosedPort1	BIT	Bit 1.2: Port 1 is closed
CommunicationPort1	BIT	Bit 1.3: Communication at port 1
LoopClosedPort2	BIT	Bit 1.4: Port 2 is closed
CommunicationPort2	BIT	Bit 1.5: Communication at port 2
LoopClosedPort3	BIT	Bit 1.6: Port 3 is closed
CommunicationPort3	BIT	Bit 1.7: Communication at port 3

Tab. 9-58: Elements of the IL_ECAT_DL_STATUS structure

9.7.10 IL_ECAT_ESC_FEATURES

Brief description The structure IL_ECAT_ESC_FEATURES describes the properties of an EtherCAT slave controller (ESC register 0x0008:0x0009).

Structural element	Data type	Description
FmmuOperationByte	BIT	Bit 0.0: Field bus memory management unit (FALSE = Bit-oriented, TRUE = Byte-oriented)
Reserved01	BIT	Bit 0.1: Reserved
DcSupport	BIT	Bit 0.2: Distributed Clocks is supported

Structural element	Data type	Description
Dc64Support	BIT	Bit 0.3: Distributed Clocks 64 bits is supported
LowJitterEbus	BIT	Bit 0.4: Jitter minimized
EnhancedLinkDetectionEbus	BIT	Bit 0.5: Enhanced link detection EBUS
EnhancedLinkDetectionMii	BIT	Bit 0.6: Enhanced link detection MII
SeparateHandlingFcsErrors	BIT	Bit 0.7: FCS errors are analyzed individually
EnhancedDcSyncActivation	BIT	Bit 1.0: Enhanced DC sync activation
LrwCommandNotSupported	BIT	Bit 1.1: Command LRW is not supported
RwCommandNotSupported	BIT	Bit 1.2: Commands BRW/APRW/FPRW are not supported
FixedFmmuSyncManConfig	BIT	Bit 1.3: Fixed FMMU/SyncManager configuration
Reserved14	BIT	Bit 1.4: Reserved
Reserved15	BIT	Bit 1.5: Reserved
Reserved16	BIT	Bit 1.6: Reserved
Reserved17	BIT	Bit 1.7: Reserved

Tab. 9-59: Elements of the IL_ECAT_ESC_FEATURES structure

9.7.11 IL_ECAT_MASTER_BUS_STATISTIC

Brief description The structure IL_ECAT_MASTER_BUS_STATISTIC describes the statistic diagnostic information on the bus communication of the EtherCAT master.

Structural element	Data type	Description
Res1	WORD	SI1: Reserved
ConfigCheckSum	DWORD	SI2: Configuration checksum (CRC32)
NumOnlineSlaves	UDINT	SI3: Number of connected slaves
NumOnlineDcSlaves	UDINT	SI4: Number of connected slaves supporting DC
NumConfigSlaves	UDINT	SI5: Number of configured slaves
NumConfigMbxSlaves	UDINT	SI6: Number of configured slaves supporting the mailbox
CntTxFrames	UDINT	SI7: Counter of sent frames
CntRxFrames	UDINT	SI8: Counter of received frames
CntLostFrames	UDINT	SI9: Counter of lost frames
CntCyclicFrames	UDINT	SI10: Counter of cyclic frames
CntCyclicDatagrams	UDINT	SI11: Counter of cyclic datagrams
CntAcyclicFrames	UDINT	SI12: Counter of acyclic frames
CntAcyclicDatagrams	UDINT	SI13: Counter of acyclic datagrams
ClearCounters	DWORD	SI14: [R/W] Reset of the frame counter and the datagram counter (1 = deletes all counters)

Tab. 9-60: Elements of the IL_ECAT_MASTER_BUS_STATISTIC structure

9.7.12 IL_ECAT_MASTER_STATE

Brief description The structure IL_ECAT_MASTER_STATE describes a summary of diagnostic information of the EtherCAT master.

Structural element	Data type	Description
NoEthernetLink	BIT	Bit 0.0: No master Ethernet link
NotInStateOp	BIT	Bit 0.1: Master is not in the state "Operational"
NotInReqState	BIT	Bit 0.2: Master is not in the requested state
Res03	BIT	Bit 0.3: Reserved
StateInit	BIT	Bit 0.4: Master is in the state "Init"
StatePreOp	BIT	Bit 0.5: Master is in the state "Pre-Operational"
StateSafeOp	BIT	Bit 0.6: Master is in the state "Safe-Operational"
Res07	BIT	Bit 0.7: Reserved
TopologyNotOk	BIT	Bit 1.0: Topology is not OK: The configured slaves do not match the online slaves, e.g. slaves are missing or the configuration of the topology does not match the real topology at the bus
SlavesNotInMasterState	BIT	Bit 1.1: One or multiple slaves are not in the master state
SlavesError	BIT	Bit 1.2: One or multiple slaves report(s) an error
Res13	BIT	Bit 1.3: Reserved
DcNotInSync	BIT	Bit 1.4: DC synchronization not within the configured limit values (only if DC is enabled)
Res15	BIT	Bit 1.5: Reserved
Res16	BIT	Bit 1.6: Reserved
Res17	BIT	Bit 1.7: Reserved
Res20 … Res37	BIT	Bit 2.0: Reserved … Bit 3.7: Reserved

Tab. 9-61: Elements of the IL_ECAT_MASTER_STATE structure

9.7.13 IL_ECAT_MBX_CONFIG_PROTOCOLS

Brief description The structure IL_ECAT_MBX_CONFIG_PROTOCOLS describes the supported mailbox protocols of a configured slave device.

Structural element	Data type	Description
MbxProtocolAoE	BIT	Bit 0.0: ADS over EtherCAT (Automation Device Specification)
MbxProtocolEoE	BIT	Bit 0.1: Ethernet over EtherCAT
MbxProtocolCoE	BIT	Bit 0.2: CAN application protocol over EtherCAT
MbxProtocolFoE	BIT	Bit 0.3: File access over EtherCAT
MbxProtocolSoE	BIT	Bit 0.4: Servo drive profile over EtherCAT
MbxProtocolVoE	BIT	Bit 0.5: Vendor-specific profile over EtherCAT
Reserved06 … Reserved37	BIT … BIT	Bit 0.6: Reserved … Bit 3.7: Reserved

Tab. 9-62: Elements of the IL_ECAT_MBX_CONFIG_PROTOCOLS structure

9.7.14 IL_ECAT_MBX_ONLINE_PROTOCOLS

Brief description The structure IL_ECAT_MBX_ONLINE_PROTOCOLS describes the supported mailbox protocols of a slave device connected to the bus.

Structural element	Data type	Description
MbxProtocolAoE	BIT	Bit 0.0: ADS over EtherCAT (Automation Device Specification)
MbxProtocolEoE	BIT	Bit 0.1: Ethernet over EtherCAT
MbxProtocolCoE	BIT	Bit 0.2: CAN application protocol over EtherCAT
MbxProtocolFoE	BIT	Bit 0.3: File access over EtherCAT
MbxProtocolSoE	BIT	Bit 0.4: Servo drive profile over EtherCAT
MbxProtocolVoE	BIT	Bit 0.5: Vendor-specific profile over EtherCAT
Reserved06 … Reserved17	BIT … BIT	Bit 0.6: Reserved … Bit 1.7: Reserved

Tab. 9-63: Elements of the IL_ECAT_MBX_ONLINE_PROTOCOLS structure

9.7.15 IL_ECAT_MBX_SIZE_INFO

Brief description The structure IL_ECAT_MBX_SIZE_INFO describes the mailbox size of a slave device.

Structural element	Data type	Description
SizeIn	UDINT	Mailbox input size (receiving slave)
SizeOut	UDINT	Mailbox output size (sending slave)

Tab. 9-64: Elements of the IL_ECAT_MBX_SIZE_INFO structure

9.7.16 IL_ECAT_MEMORY_INFO

Brief description The structure IL_ECAT_MEMORY_INFO describes the storage area.

Structural element	Data type	Description
Offset	UDINT	Memory offset
Size	UDINT	Memory size

Tab. 9-65: Elements of the IL_ECAT_MEMORY_INFO structure

9.7.17 IL_ECAT_PORT_ERROR_COUNTER

Brief description The structure IL_ECAT_PORT_ERROR_COUNTER describes the error counter of a port.

Structural element	Data type	Description
InvalidFrameCounter	BYTE	Error counter of invalid frames received (Register 0x0300/0x0302/0x0304/0x0306)
RxErrorCounter	BYTE	Error counter of invalid frames received (MII/EBUS) (Register 0x0301/0x0303/0x0305/0x0307)

Structural element	Data type	Description
FwdRxErrorCounter	BYTE	Error counter of invalid frames forwarded (register 0x0308:0x030B)
LostLinkCounter	BYTE	Error counter of link loss (register 0x0310:0x0313), Only if loop detection "Auto" and if the port is "open".

Tab. 9-66: Elements of the IL_ECAT_PORT_ERROR_COUNTER structure

9.7.18 IL_ECAT_PORT_STATE

Brief description The structure IL_ECAT_PORT_STATE describes the EtherCAT port status of a slave device.

Compared to the DL status, only the status of the physical ports is shown.

Structural element	Data type	Description
ConnectionPort0	BIT	Bit 0.0: Connection present at port 0
ConnectionPort1	BIT	Bit 0.1: Connection present at port 1
ConnectionPort2	BIT	Bit 0.2: Connection present at port 2
ConnectionPort3	BIT	Bit 0.3: Connection present at port 3
LinkPort0	BIT	Bit 0.4: Link available at port 0
LinkPort1	BIT	Bit 0.5: Link available at port 1
LinkPort2	BIT	Bit 0.6: Link available at port 2
LinkPort3	BIT	Bit 0.7: Link available at port 3
LoopClosedPort0	BIT	Bit 1.0: Port 0 is closed (only physical ports)
LoopClosedPort1	BIT	Bit 1.1: Port 1 is closed (only physical ports)
LoopClosedPort2	BIT	Bit 1.2: Port 2 is closed (only physical ports)
LoopClosedPort3	BIT	Bit 1.3: Port 3 is closed (only physical ports)
CommunicationPort0	BIT	Bit 1.4: Communication at port 0
CommunicationPort1	BIT	Bit 1.5: Communication at port 1
CommunicationPort2	BIT	Bit 1.6: Communication at port 2
CommunicationPort3	BIT	Bit 1.7: Communication at port 3

Tab. 9-67: Elements of the IL_ECAT_PORT_STATE structure

The diagnostic bit "Connection(PortX)" is a logical linking of other status bits:

Communication	LoopClosed	LinkStatus	=> Connection	Note
0	0	0	0	E.g. port not configured
0	0	1	1	-
0	1	0	0	Standard "Loop closed"
0	1	1	0	Maybe "old" devices
1	0	0	1	Maybe "old" devices
1	0	1	1	Standard "Loop open"

Communication	LoopClosed	LinkStatus	=> Connection	Note
1	1	0	0	-
1	1	1	0	E.g. EK1122 "closed port"

9.7.19 IL_ECAT_SLAVE_CONFIG_INFO

Brief description The structure IL_ECAT_SLAVE_CONFIG_INFO describes information on a slave device configured in the EtherCAT master.

Structural element	Data type	Description
AutoIncAddr	UINT	Auto-increment address
EthercatAddr	UINT	EtherCAT address ("fixed physical address")
IdentifyAdo	UINT	Address offset (ADO) for identification command
IdentifyValue	UINT	Identification value
SlaveHandle	DWORD	Internal SlaveID (internal use)
HcGroupIdx	DWORD	Index of the Hot Connect Group (0 = mandatory)
PreviousEthercatAddr	UINT	EtherCAT address of the previous slave (in the topology)
PreviousPort	UINT	Port of the previous slave (in the topology)
SlaveIdentity	IL_ECAT_SLAVE_IDENTITY_INFO	Slave identity information
SlaveName	STRING(127)	Configured device name
MbxProtocols	IL_ECAT_MBX_CONFIG_PROTOCOLS	Supported mailbox protocols
MbxStandard	IL_ECAT_MBX_SIZE_INFO	Mailbox default size
MbxBootstrap	IL_ECAT_MBX_SIZE_INFO	Mailbox bootstrap size
ProcessDataIn	ARRAY[0..3] OF IL_ECAT_MEMORY_INFO	Process data range of the inputs
ProcessDataOut	ARRAY[0..3] OF IL_ECAT_MEMORY_INFO	Process data range of the outputs
NumProcessVarsIn	UINT	Number of input process data of variables
NumProcessVarsOut	UINT	Number of output process data of variables
PortDescriptor	BYTE	Port descriptor (0b10 = EBUS, 0b11 = MII/…) (Port0^Bit0:1, Port1^Bit2:3, etc.)
Reserved01	ARRAY[0..2] OF BYTE	Reserved
WkcStateDiagOffsIn	ARRAY[0..3] OF UINT	Internal (internal use)
WkcStateDiagOffsOut	ARRAY[0..3] OF UINT	Internal (internal use)
Reserved02	ARRAY[0..2] OF UDINT	Reserved
IsPresent	BOOL	Slave is currently connected to the bus (online)
IsHcGroupPresent	BOOL	Hot Connect Group of the slave is currently connected to the bus
DcSupport	BOOL	Slave supports DC (Distributed Clocks)

Tab. 9-68: Elements of the IL_ECAT_SLAVE_CONFIG_INFO structure

9.7.20 IL_ECAT_SLAVE_ERROR_COUNTER

Brief description The structure IL_ECAT_SLAVE_ERROR_COUNTER describes the error counter of a slave device.

Structural element	Data type	Description
ProcUnitErrorCounter	BYTE	Error counter "Processing unit" (e.g. FCS) (Register 0x030C)
PdiErrorCounter	BYTE	Error counter "Process Data Interface" (PDI) (Register 0x030D)
PortErrorCounter	ARRAY[0..3] OF IL_ECAT_PORT_ERROR_COUNTER	Error counter per port

Tab. 9-69: Elements of the IL_ECAT_SLAVE_ERROR_COUNTER structure

9.7.21 IL_ECAT_SLAVE_IDENTITY_INFO

Brief description The structure IL_ECAT_SLAVE_IDENTITY_INFO describes identity data of an EtherCAT slave device.

Structural element	Data type	Description
VendorId	UDINT	Vendor ID (e.g. Rexroth = 16#24)
ProductCode	UDINT	Product code (vendor-specific)
RevisionNumber	UDINT	Revision number
SerialNumber	UDINT	Serial number

Tab. 9-70: Elements of the IL_ECAT_SLAVE_IDENTITY_INFO structure

9.7.22 IL_ECAT_SLAVE_ONLINE_INFO

Brief description The structure IL_ECAT_SLAVE_ONLINE_INFO describes "online" information on a slave device connected to the EtherCAT master.

Structural element	Data type	Description
AutoIncAddr	UINT	Auto-increment address
EthercatAddr	UINT	EtherCAT address ("fixed physical address") (ESC register 0x0010)
StationAlias	UINT	Station alias ("second slave address") (ESC register 0x0012)
IdentifyValue	UINT	Identification value (only if an ADO is configured, refer to IL_ECAT_SLAVE_CONFIG_INFO → IdentifyAdo)
SlaveHandle	DWORD	Internal SlaveID (internal use)
PortSlaveHandles	ARRAY[0..3] OF DWORD	Port-internal SlaveIDs (internal use)
SlaveIdentity	IL_ECAT_SLAVE_IDENTITY_INFO	Slave identity information (EEPROM offset 0x0008)
EscType	BYTE	EtherCAT slave controller type (ESC register 0x0000)
EscRevision	BYTE	EtherCAT slave controller revision number (ESC register 0x0001)

Structural element	Data type	Description
EscBuild	WORD	EtherCAT slave controller build number (ESC register 0x0002)
EscFeatures	IL_ECAT_ESC_FEATURES	EtherCAT slave controller-supported features (ESC register 0x0008)
PortDescriptor	BYTE	Port descriptor (ESC register 0x0007) (0b10 = EBUS, 0b11 = MII/…) (Port0^Bit0:1, Port1^Bit2:3, etc.)
Reserved01	BYTE	Reserved
AlStatus	WORD	Application layer status (ESC register 0x0130)
AlStatusCode	WORD	Application layer status code (ESC register 0x0134) (is reset in the master if the state of the slave changes)
MbxProtocols	IL_ECAT_MBX_ON-LINE_PROTOCOLS	Supported mailbox protocols (EEPROM offset 0x001C); This value is "0" if the slave is configured in the master, refer to IL_ECAT_SLAVE_CONFIG_INFO.
DlStatus	IL_ECAT_DL_STATUS	Data link status (ESC register 0x0110)
PortState	IL_ECAT_PORT_STATE	Port status
PreviousPort	UINT	Port of the previous slave (in the topology)
SystemTimeDifference	UDINT	System time difference in ns (ESC register 0x092C) (this value is only valid if DC is configured)
SlaveDelay	UDINT	Time delay "after" the slave in ns (this value is only valid if DC is configured)
PropagationDelay	UDINT	Transmission delay in ns (ESC Register 0x0928) (this value is only valid if DC is configured)
Reserved02	ARRAY[0..3] OF DWORD	Reserved
DcSupport	BOOL	Slave supports DC (Distributed Clocks)
Dc64Support	BOOL	Slave supports 64 bit DC (Distributed Clocks)
IsRefClock	BOOL	Slave is reference clock
LineCrossed	BOOL	EtherCAT cabling of the slave is defective (IN-OUT port exchanged)

Tab. 9-71: Elements of the IL_ECAT_SLAVE_ONLINE_INFO structure

9.7.23 IL_ECAT_SLAVE_STATE

Brief description The structure IL_ECAT_SLAVE_STATE describes diagnostic status information on a slave devices.

Structural element	Data type	Description
NotOnline	BIT	Bit 0.0: Slave is not connected to the bus
NotInStateOp	BIT	Bit 0.1: Slave is not in the state "Operational"
Error	BIT	Bit 0.2: Slave reports errors, refer to "AL status code"
Res03	BIT	Bit 0.3: Reserved

Structural element	Data type	Description
StateInit	BIT	Bit 0.4: Slave is in the state "Init"
StatePreOp	BIT	Bit 0.5: Slave is in the state "Pre-Operational"
StateSafeOp	BIT	Bit 0.6: Slave is not in the state "Safe-Operational"
Res07	BIT	Bit 0.7: Reserved
Res10 … Res37	BIT ... BIT	Bit 1.0: Reserved … Bit 3.7: Reserved

Tab. 9-72: Elements of the IL_ECAT_SLAVE_STATE structure

9.7.24 IL_ECAT_SOE_ATTRIBUTE

Brief description The structure IL_ECAT_SOE_ATTRIBUTE describes the SoE parameter attribute (refer to chapter 9.6.2 "IL_ECATSoeAttributeDecoding" on page 301).

Structural element	Data type	Description
Res31	BOOL	Bit 31: Reserved
WriteProtectedCp4	BOOL	Bit 30: Parameter is write-protected in the state OP (CP4)
WriteProtectedCp3	BOOL	Bit 29: Parameter is write-protected in the state SafeOP (CP3)
WriteProtectedCp2	BOOL	Bit 28: Parameter is write-protected in the state PreOP (CP2)
DecPoints	USINT	Bit 24-27: Number of decimal places
Res23	BOOL	Bit 23: Reserved
DataType	IL_ECAT_SOE_DATA_TYPE	Bit 20-22: Data type and display format
IsCommand	BOOL	Bit 19: Parameter is a command
DataLength	IL_ECAT_SOE_DATA_LENGTH	Bit 16-18: Data length
ConversionFactor	UINT	Bit 0-15: Factor for representation

Tab. 9-73: Elements of the IL_ECAT_SOE_ATTRIBUTE structure

9.7.25 IL_ECAT_SOE_DATA_LENGTH

Brief description The enumeration type IL_ECAT_SOE_DATA_LENGTH supports the evaluation of the SoE parameter attribute with regard to data length and differentiation between list parameter (VAR_x_BYTE) and individual parameter (FIX_x_BYTE) (refer to chapter 9.6.2 "IL_ECATSoeAttributeDecoding" on page 301).

Name	Value	Description
SOE_DATALENGTH_RES0	0	Reserved
SOE_DATALENGTH_FIX_2_BYTE	1	Data length is 2 bytes
SOE_DATALENGTH_FIX_4_BYTE	2	Data length is 4 bytes
SOE_DATALENGTH_FIX_8_BYTE	3	Data length is 8 bytes
SOE_DATALENGTH_VAR_1_BYTE	4	Data length is variable with 1 byte of data

Name	Value	Description
SOE_DATALENGTH_VAR_2_BYTE	5	Data length is variable with 2 bytes of data
SOE_DATALENGTH_VAR_4_BYTE	6	Data length is variable with 4 bytes of data
SOE_DATALENGTH_VAR_8_BYTE	7	Data length is variable with 8 bytes of data

Tab. 9-74: Elements of the IL_ECAT_SOE_DATA_LENGTH enumeration type

9.7.26 IL_ECAT_SOE_DATA_STATUS

Brief description The enumeration type IL_ECAT_SOE_DATA_STATUS describes the data status of an SoE parameter (e.g. bits 0-3 are used for the status of a command execution).

Name	Value	Description
SOE_DATA_CMD_NOT_SET	0	Command is not yet enabled or it was canceled
SOE_DATA_CMD_SET	1	Command was enabled
SOE_DATA_CMD_END_OK	3	Command execution was completed successfully
SOE_DATA_CMD_INTERRUPTED	5	Command execution was interrupted
SOE_DATA_CMD_RUNNING	7	Command is enabled, but not yet completed
SOE_DATA_CMD_ERROR	15	Error: Command execution not possible
SOE_DATA_DATUM_INVALID	256	(Bit 8) data is invalid

Tab. 9-75: Elements of the IL_ECAT_SOE_DATA_STATUS enumeration type

9.7.27 IL_ECAT_SOE_DATA_TYPE

Brief description The enumeration type IL_ECAT_SOE_DATA_TYPE supports the evaluation of the SoE parameter attribute with regard to data type and display format (refer to chapter 9.6.2 "IL_ECATSoeAttributeDecoding" on page 301).

Name	Value	Description
SOE_DATATYPE_BINARY	0	Binary number (representation binary)
SOE_DATATYPE_UDEC	1	Integer without sign (decimal representation)
SOE_DATATYPE_DEC	2	Integer with sign (decimal representation)
SOE_DATATYPE_HEX	3	Integer without sign (hexadecimal representation)
SOE_DATATYPE_TEXT	4	Extended character set (text representation)
SOE_DATATYPE_IDN	5	Integer without sign (IDN representation, e.g. "S-0-0000")
SOE_DATATYPE_FLOAT	6	Floating-point number (decimal representation with exponent)
SOE_DATATYPE_RES7	7	Reserved

Tab. 9-76: Elements of the IL_ECAT_SOE_DATA_TYPE enumeration type

9.7.28 IL_ECAT_SOE_ELEMENT

Brief description The enumeration type IL_ECAT_SOE_ELEMENT describes the elements of an SoE parameter.

Name	Value	Description
SOE_ELEMENT_STATUS	16#01	Service channel of data status
SOE_ELEMENT_NAME	16#02	Name (max. 64 bytes) (optional)
SOE_ELEMENT_ATTRIBUTE	16#04	Attribute
SOE_ELEMENT_UNIT	16#08	Unit (optional)
SOE_ELEMENT_MIN	16#10	Minimum value (optional)
SOE_ELEMENT_MAX	16#20	Maximum value (optional)
SOE_ELEMENT_OPDATA	16#40	Operating data
SOE_ELEMENT_DEFAULT	16#80	Default value (optional)

Tab. 9-77: Elements of the IL_ECAT_SOE_ELEMENT enumeration type

9.7.29 IL_ECAT_SOE_PARAM_TYPE

Brief description The enumeration type IL_ECAT_SOE_PARAM_TYPE distinguishes between SoE standard parameters and product-specific parameters.

Name	Value	Description
SOE_S_PARAM	0	Standard IDN ("S-parameters")
SOE_P_PARAM	1	Product-specific IDN ("P-parameters")

Tab. 9-78: Elements of the IL_ECAT_SOE_PARAM_TYPE enumeration type

9.7.30 IL_ECAT_STATE

Brief description The enumeration type IL_ECAT_STATE describes the EtherCAT state.

Name	Value	Description
ECAT_STATE_UNKNOWN	0	EtherCAT state "Unknown"
ECAT_STATE_INIT	1	EtherCAT state "Init"
ECAT_STATE_PREOP	2	EtherCAT state "Pre-Operational"
ECAT_STATE_BOOTSTRAP	3	EtherCAT state "Bootstrap" (only for slaves) [4)]
ECAT_STATE_SAFEOP	4	EtherCAT state "Safe-Operational"
ECAT_STATE_OP	8	EtherCAT state "Operational"

Tab. 9-79: Elements of the IL_ECAT_STATE enumeration type

9.8 Error codes

9.8.1 EtherCAT error table

All error codes of the EtherCAT master are part of the MLC diagnostic system on the MLC control system:

- The category is generally "F0" (= non-fatal error)
- The EtherCAT Master currently uses the diagnostic groups 0xEC, 0xED, 0xEE and 0xEF
- Subsequently, there is a 4-digit error number

4) *Optional state for slave devices are mainly for FW update; can only be reached from the slave state "Init".*

Refer to the error descriptions under MLC diagnostics.



For the error code, go to the "Index" search in the IndraWorks online help, e.g. "F0EC0001".

For the function blocks, the error code is displayed at the output "ErrorIdent.Additional1".

For specific errors of the function blocks, refer to the respective functional description.

The EtherCAT error codes are also listed as GVL "ErrorCodes" in the RIL_Ethercat library.

10 RIL_MappingList.library

10.1 Introduction and overview

To allow acyclic access to field bus slaves (DP slave, PNIO device and ENIP adapter), a mapping table is implemented into the field bus slaves.

A mapping to data objects of the control is stored in this mapping table to allow bus-specific acyclic field bus access.

This mapping (= addressing) rule is executed for an acyclic field bus access.

It is planned to create the mapping table on the user interface via a configuration tool and to load it to the control.

As long as this configuration tool is not available, the mapping table can be filled with the RIL_MappingList.library from the IEC program and edited.

In the mapping table, PLC addresses to access user data are assigned to field bus-specific (object) accesses.

These addresses can be:

- symbolic PLC operands
- absolute memory addresses

The mapping table is filled either with contents of the I/O configuration (final configuration, depending on the mapping configurator in the interface) or of the IEC program via the function block calls described.

The mapping table exists once for each field bus slave.

The field bus addresses can be freely used within the value ranges allowed by the particular field bus.

The number of possible entries in the mapping table is presently limited to 256.

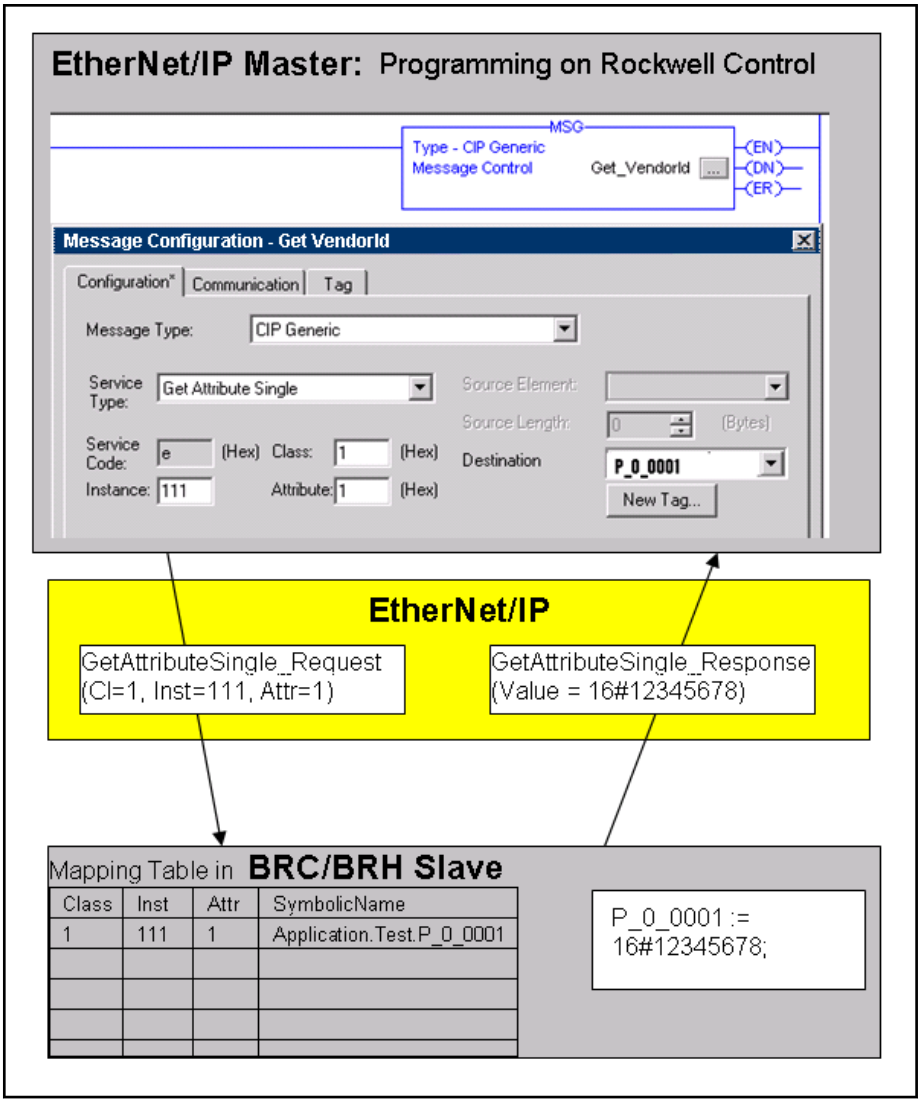


Fig. 10-1: Example access via Ethernet/IP

Areas of application and function blocks of the RIL_MappingList.library

Area of application (folder in library)	
Function blocks	**Description**
POUs	
IL_SlaveMapListInit, page 323	Initializing and deleting the mapping table
IL_SlaveMapListAddEntry, page 324	Adding an entry to the mapping table
Data types (DataTypes):	
IL_FIELDBUSTYPE, page 328	Selecting the type of the particular field bus
IL_FIELDBUSOBJECT, page 327	Containing the addressing of the individual field bus objects
IL_ADDRESSTYPE, page 327	Selecting the type of the variables to be mapped

Tab. 10-1: RIL_MappingList library

Selecting the slave

The enumeration type IL_BUSSLAVE of the RIL_FieldbusTypes.library is used to select the respective field bus slave.

Also refer to: IL_BUSSLAVE, page329

10.2 POUs

10.2.1 IL_SlaveMapListInit

Brief description

This function block initializes and deletes the entire mapping table.

This ensures that there are no history entries.

If the RIL_MappingList library is used, the list should be initialized before access entries are added, to avoid interactions with other methods for creating access entries.

This function block can likewise be used to disable outside access via the field bus again.

Target system	Library/folder
MLC	RIL_MappingList/ POUs
MTX	RIL_MappingList/ POUs

Tab. 10-2: Reference table

Interface description

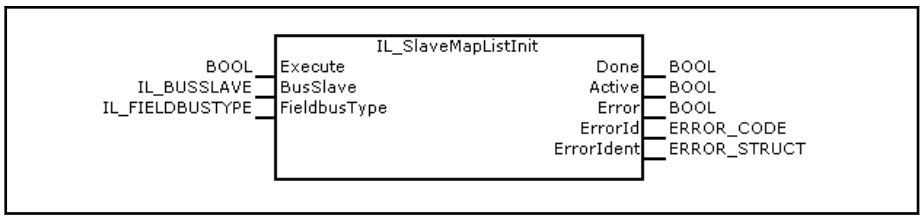


Fig. 10-2: Interface of the IL_SlaveMapListInit function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the field bus slave in the order of the configuration tree. If only one slave/device/adapter is available, the value of this slave/device/adapter is always IL_BUSSLAVE_0. IL_BUSSLAVE, page 329
	FieldbusType	IL_FIELDBUSTYPE	Selects the field bus type (Profinet IO, Profibus DP, EtherNet/IP) Also refer to IL_FIELDBUSTYPE on page 328
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable NO_TABLE_USED)

Tab. 10-3: IL_SlaveMapListInit I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Execute
FieldbusType	IL_FIELDBUS-TYPE	IL_DP_SLAVE	IL_ENIP_ADAPT-ER	IL_DP_SLAVE	Rising edge at Execute

Tab. 10-4: Minimum, maximum and default values of the IL_SlaveMapListInit inputs

Functional description

Function block IL_SlaveMapListInit serves to delete an existing mapping list.

A mapping list may have been created through the following actions:

- The mapping table was generated via the user interface.
- The mapping table was generated earlier by the PLC program.

The FB must be called in the following cases:

- if the mapping table is not generated via the user interface,
- if it is intended to remove entries from the mapping table (i.e., complete deletion of the table).

Troubleshooting

The function block uses error table **NO_TABLE_USED 16#0000**. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	16#0000 16#0001	BusSlave not in configuration table
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
INPUT_RANGE_ERROR	16#0108	0	Field bus type not supported

Tab. 10-5: Error codes of the IL_SlaveMapListInit function block

10.2.2 IL_SlaveMapListAddEntry

Brief description

The function block adds an entry to the mapping table. The entry is permanently associated with the field bus object address. A field bus object address can be assigned only once.

Target system	Library/folder
MLC	RIL_MappingList/ POUs
MTX	RIL_MappingList/ POUs

Tab. 10-6: Reference table

Interface description

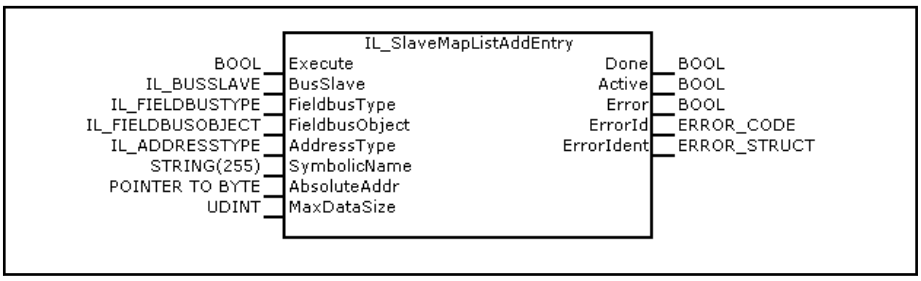


Fig. 10-3: FB IL_SlaveMapListAddEntry interface

	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusSlave	IL_BUSSLAVE	Instance of the field bus slave in the order of the configuration tree. If only one slave/device/adapter is available, the value of this slave/device/adapter is always IL_BUSSLAVE_0. IL_BUSSLAVE, page 329
	FieldbusType	IL_FIELDBUSTYPE	Selects the field bus type (Profinet IO, Profibus DP, EtherNet/IP) Also refer to IL_FIELDBUSTYPE on page 328
	FieldbusObject	IL_FIELDBUSOB-JECT	Field bus-specific object address Also refer to IL_FIELDBUSOBJECT on page 327
	AddressType	IL_ADDRESSTYPE	Addressing type Also refer to IL_ADDRESSTYPE on page 327
	SymbolicName	STRING(255)	Symbolic name of a variable from the IEC program. The symbol must comply with pertinent IEC programming rules. To ensure uniqueness, task and POU must perhaps also be specified.
	AbsoluteAddr	POINTER TO BYTE	Pointer to an absolutely addressed data area
	MaxDataSize	UDINT	Maximum allowed byte length for access to the object. This parameter can be used to limit the read/write access to the number of bytes specified. If the input address type = FBT_SYMBOL_IEC (_RO), this limit may also be below the size of the object addressed via Symbolic-Name.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error (from RIL_CommonTypes)
	ErrorIdent	ERROR_STRUCT	Detailed diagnosis (ErrorTable NO_TABLE_USED)

Tab. 10-7: IL_SlaveMapListAddEntry I/O interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusSlave	IL_BUSSLAVE	IL_BUSSLAVE_0	IL_BUSSLAVE_4	IL_BUSSLAVE_0	Rising edge at Execute
FieldbusType	IL_FIELDBUS-TYPE	IL_DP_SLAVE	IL_ENIP_ADAPTER	IL_DP_SLAVE	Rising edge at Execute
FieldbusObject	IL_FIELDBU-SOBJECT	Depending on the field bus	Depending on the field bus		Rising edge at Execute
AddressType	IL_ADRES-STYP	FBT_SYMBOL_IEC	FBT_ADR_IEC_RO	FBT_SYM-BOL_IEC	Rising edge at Execute
SymbolicName	STRING(255)	n.a.	n.a.	’’	Rising edge at Execute

Name	Type	Min. value	Max. value	Default value	Effective
AbsoluteAddr	POINTER TO BYTE	n.a.	n.a.	0	Rising edge at Execute
MaxDataSize	UDINT	0	Depending on the field bus	0	Rising edge at Execute

Tab. 10-8: Minimum, maximum and default values of the IL_SlaveMapListAddEntry

Functional description

Example: Entry of the symbolic read access to IEC variable 'wrvar' via field bus index 16#1001 to slot and subslot 1 for a PN device. Write access is disabled. Symbol 'wrvar' must be entered in the symbol table.

Extract from the program

```
fbMapAdd: IL_SlaveMapListAddEntry;
wrvar:    DINT;

fbMapAdd.FieldbusType:=           IL_FIELDBUSTYPE.IL_PNIO_DEVICE;
fbMapAdd.BusSlave:=               IL_BUSSLAVE_0;
fbMapAdd.FieldbusObject.Index:=   16#1001;
fbMapAdd.FieldbusObject.Slot:=    1;
fbMapAdd.FieldbusObject.SubSlot:= 1;
fbMapAdd.MaxDataSize:=            SIZEOF(wrvar);
fbMapAdd.AddressType:=  IL_ADDRESSTYPE.IL_SYMBOL_IEC_RO; //read only
fbMapAdd.SymbolicName:=           'Application.MappingTest.wrvar';

fbMapAdd  (Execute:= TRUE);
```

Example: Entry of the absolutely addressed read and write access to IEC variable 'wrvar' via field bus index 16#1002 to slot and subslot 1 for a PN device.

Extract from the program

```
fbMapAdd: IL_SlaveMapListAddEntry;
wrvar:    DINT;

fbMapAdd.Execute:=                TRUE;
fbMapAdd.FieldbusType:=           IL_FIELDBUSTYPE.IL_PNIO_DEVICE;
fbMapAdd.FieldbusObject.Index:=   16#1002;
fbMapAdd.FieldbusObject.Slot:=    1;
fbMapAdd.FieldbusObject.SubSlot:= 1;
fbMapAdd.MaxDataSize:=            SIZEOF(wrvar);
fbMapAdd.AddressType:=            IL_ADDRESSTYPE.IL_ADDR_IEC;
fbMapAdd.AbsolutAddr:=            ADR(wrvar);

fbMapAdd  (Execute:= TRUE);
```

Troubleshooting

The function block uses error table **NO_TABLE_USED 16#0000**. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	
SYSTEM_ERROR	16#0101	16#0000 16#0001	BusSlave not in configuration table
STATE_MACHINE_ERROR	16#0105	0	Error in the state machine
INPUT_RANGE_ERROR	16#0108	0	Field bus type not supported
INPUT_RANGE_ERROR	16#0109	0	Address type not supported
SYSTEM_ERROR	16#010A	0	No entry more available anymore.
SYSTEM_ERROR	16#010B	0	Entry already in use
INPUT_RANGE_ERROR	16#010C	0	Symbol not found in SymbolicName
INPUT_RANGE_ERROR	16#010D	0	FieldbusObject invalid
INPUT_RANGE_ERROR	16#010E	0	FieldbusObject reserved

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#010F	0	AbsoluteAddr invalid
INPUT_RANGE_ERROR	16#0110	0	MaxDataSize is zero
INPUT_RANGE_ERROR	16#0111	0	MaxDataSize is too high
INPUT_RANGE_ERROR	16#0112	0	Index already reserved
INPUT_RANGE_ERROR	16#0113	0	Slot already reserved
INPUT_RANGE_ERROR	16#0114	0	Subslot already reserved
INPUT_RANGE_ERROR	16#0115	0	Class already reserved
INPUT_RANGE_ERROR	16#0116	0	Instance already reserved
INPUT_RANGE_ERROR	16#0117	0	Attributes already reserved

Tab. 10-9: Error codes of the IL_SlaveMapListAddEntry function block

10.3 Data types (DataTypes)

10.3.1 IL_ADDRESSTYPE

Brief description Enumeration type IL_ADDRESSTYPE serves to select the type of the variables to be mapped.

Element	Value	Description
IL_SYMBOL_IEC	0	Symbolic IEC variable name; parameter SymbolicName is evaluated. Read and write access allowed
IL_ADDR_IEC	1	Absolute IEC address; parameter AbsoluteAddr is evaluated. Read and write access allowed
IL_SYMBOL_IEC_RO	2	Symbolic IEC variable name; parameter SymbolicName is evaluated. Read access allowed only
IL_ADDR_IEC_RO	3	Absolute IEC address; parameter AbsoluteAddr is evaluated. Read access allowed only

Tab. 10-10: Elements of enumeration type IL_ADDRESSTYPE

10.3.2 IL_FIELDBUSOBJECT

Brief description Data structure IL_FIELDBUSOBJECT contains the addressing of the various field bus objects.

	Type	Required for ProfibusDP IL_DP_SLAVE	Required for ProfinetIO IL_PNIO_DEVICE	Required for EthernetIP IL_ENIP_ADAPTER	Description
Slot	WORD	Yes	Yes		Slot number
Index	WORD	Yes	Yes		Index
SubSlot	WORD		Yes		Subslot number
Class	WORD			Yes	Class

	Type	Required for ProfibusDP IL_DP_SLAVE	Required for ProfinetIO IL_PNIO_DEVICE	Required for EthernetIP IL_ENIP_ADAPTER	Description
Instance	WORD			Yes	Instance
Attributes	WORD			Yes	Attribute

Tab. 10-11: Interface description of data structure IL_FIELDBUSOBJECT

Addressing of the field bus objects for ProfibusDP:

	Type	Min. value	Max. value	Default value	Description
Slot	WORD	0	254		Slot number
Index	WORD	0	254		Index

Tab. 10-12: IL_FIELDBUSOBJECT for ProfibusDP

Addressing of the field bus objects for ProfinetIO:

	Type	Min. value	Max. value	Default value	Description
Slot	WORD	0	16#7FFF		Slot number
Index	WORD	0	16#7FFF		Index (16#8000 to 16#ffff are reserved)
SubSlot	WORD	1	16#8FFF		SubslotNummer (subslot number)

Tab. 10-13: IL_FIELDBUSOBJECT for ProfinetIO

Addressing of the field bus objects for EtherNet/IP:

	Type	Min. value	Max. value	Default value	Description
Class	WORD	0	16#FFFF		
Instance	WORD	1	16#FFFF		
Attributes	WORD	1	16#FFFF		

Tab. 10-14: IL_FIELDBUSOBJECT for EtherNet/IP

10.3.3 IL_FIELDBUSTYPE

Brief description Enumeration type IL_FIELDBUSTYPE serves to select the type of the appropriate field bus.

At present, field bus slaves are available for

- Profibus DP,
- Profinet IO,
- EtherNet/IP.

Element	Value	Description
IL_DP_SLAVE	0	Profibus DP
IL_PNIO_DEVICE	1	Profinet I/O
IL_ENIP_ADAPTER	2	EtherNet/IP

Tab. 10-15: Elements of enumeration type IL_FIELDBUSTYPE

10.3.4 IL_BUSSLAVE

Brief description

The enumeration type IL_BUSSLAVE of the RIL_FieldbusTypes.library is used to select the respective field bus slave.

In addition to the onboard real-time Ethernet interface, other controllers can also be available in a control via function modules.

The devices are distinguished based on their ascending order in the configuration. Indexes are assigned to the devices in the following order

- Onboard
- Function module 1,
- Function module 2,
- Function module 3,
- Function module 4,

If, for example, the onboard real-time Ethernet interface is a PROFINET IO controller and a PROFINET IO device is configured on FM1, the value assigned to the device on the FM1 is IL_BUSSLAVE_0.

Element	Value	Description
IL_BUSSLAVE_0	0	First device
IL_BUSSLAVE_1	1	Second device
IL_BUSSLAVE_2	2	Third device
IL_BUSSLAVE_3	3	Fourth device
IL_BUSSLAVE_4	4	Fifth device

Tab. 10-16: Elements of the IL_BUSSLAVE enumeration type

11 RIL_SercosIII.library

11.1 Introduction and overview

The “RIL_SercosIII” library is the interface between the PLC programming environment and the Sercos III I/O devices. For example, this interface allows to diagnose the Sercos III I/O devices or to acyclically exchange Sercos parameters.

Target systems

The library can be used in the following systems:

Target system	Library
MLC/MTX	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ 20VRS	RMB_SercosIII_Util

Areas of application and function blocks of RIL_SercosIII

Area of application (folder in library)	
Function blocks	**Description**
Acyclic services (AcyclicCommunication)	
IL_SIIISvcRead, page 333	Reading a parameter via the Sercos III service channel
IL_SIIISvcWrite, page 338	Writing a parameter via the Sercos III service channel
Acyclic services/parameters (AcyclicCommunication/Parameter)	
IL_SIIICommand, page 341	Executing commands of a Sercos III slaves via the Sercos III service channel
IL_SIIIReadParameter, page 348	Can read 2-byte or 4-byte parameters from a slave device via the Sercos III service channel (DINT)
IL_SIIIReadRealParameter, page 352	Can read 2-byte or 4-byte parameters from a slave device via the Sercos III service channel (REAL)
IL_SIIIReadStringParameter, page 354	Can read text lists from a slave device via the Sercos III service channel (STRING)
IL_SIIIWriteParameter, page 356	Can write 2-byte or 4-byte parameters of a slave device via the Sercos III service channel (DINT)
IL_SIIIWriteRealParameter, page 359	Can write 2-byte or 4-byte parameters of a slave device via the Sercos III service channel (REAL)
With data type:	
IL_SERCOS_DATA_STATUS, page 361	Status of a command started by the **IL_SIIICommand** function block from the RIL_SercosIII.library is displayed in plain text
Backing up and restoring (Backup_Restore)	
IL_SIIIParameterBackup, page 362	Backs up the parameters of a Sercos device in a text-based file on a memory medium of the Sercos master.
IL_SIIIParameterRestore, page 368	Imports the S-parameters and the P-parameters of a Sercos device from a text-based file
With data types:	
IL_BACKUP_OPTIONS, page 374	Possible parameter selection for the backup
IL_BACKUP_ERRORS, page 374	Error during backup/restore

Diagnostics	
IL_SIIIGetConfigSlaveAdrList, page 374	Creates a list of Sercos addresses of all configured PLC Sercos devices
IL_SIIIGetDiagSlaveAdrList, page 376	Creates a list of the Sercos addresses of all PLC Sercos devices with present diagnostic information
IL_SIIIGetLiveSlaveAdrList, page 377	Creates a list of the Sercos addresses of all Sercos devices found at the bus
IL_SIIIMasterState, page 379	Describes the status of the Sercos master
IL_SIIIMasterStateDetails, page 380	Describes the detailed status of the Sercos master with error counters and topology information
IL_SIIIRemoteSlaveState, page 381	Describes the status of a slave operated at the Sercos master
IL_SIIIRemoteSlaveStateDetails, page 383	Describes the detailed status of a slave operated at the Sercos master
With data types:	
IL_SIII_REMOTE_SLAVE_STATE_DETAILS, page 385 IL_SIII_REMOTE_SLAVE_STATE_EX, page 388 IL_SIII_REMOTE_SLAVE_STATE_EX_BITS, page 388 IL_SIII_DEVICE_STATUS, page 389 IL_SIII_DEVICE_STATUS_UNION, page 389 IL_SIII_DEVICE_STATUS_UNION_BITS, page 390 IL_SIII_TOPOLOGY_PORT_STATUS, page 390 IL_SIII_DIAGNOSTIC_NUMBER, page 391 IL_SIII_DIAGNOSTIC_NUMBER_R, page 392 IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP, page 392 IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE, page 392 IL_SIII_DIAGNOSTIC_NUMBER_CLASS, page 393 IL_SIII_DEVICE_TYPE, page 393 IL_SIII_DRIVE_DIAG, page 393 IL_SIII_DRV_ERROR, page 394 IL_SIII_DRV_ERROR_BITS, page 394 IL_SIII_DRV_WARNING, page 395 IL_SIII_DRV_WARNING_BITS, page 395 IL_SIII_DRV_MESSAGE, page 396 IL_SIII_DRV_MESSAGE_BITS, page 396 IL_SIII_IO_DIAG, page 397 IL_SIII_IO_DIAG_STATUS_CODE_TYPE, page 397 IL_SIII_IO_DIAG_STATUS_CODE, page 398	
Auxiliary functions (utilities)	
IL_SIIIAttribute, page 405	Breaking down elements of the Sercos attribute

IL_SIIIElementsToIdn, page 408	Summarizes the individual elements of an IDN to one MB_IDN value
IL_SIIIGetSercosPhase, page 409	Reads the Sercos phase of a Sercos bus master
IL_SIIISetSercosPhase, page 410	Writes the Sercos phase of a Sercos bus master
IL_SIIIStringToIdn, page 412	Converts a string such as "S-0-0001.0.0" or the shortened versions "S-0-0001" or "S1" into an MB_IDN value
With data types:	
IL_SIII_DATA_LENGTH, page 413	Supports the analysis of the Sercos attribute with regard to data length and differentiation between list parameters and individual parameters
IL_SIII_DATA_TYPE, page 414	Supports the analysis of the Sercos attribute with regard to data type and display format
IL_SIII_ELEMENT, page 414	Describes the Sercos elements of a parameter
IL_SIII_PARAM_TYPE, page 414	Distinguishes between standard IDN and product-specific IDN
IL_SIII_SERCOS_PHASE, page 415	Contains the possible active Sercos phases

Tab. 11-1: RIL_SercosIII library

Selecting the master

The **IL_BUSMASTER** enumeration type of the RIL_FieldbusTypes.library is used to select the respective Sercos III master.

Currently, only the onboard master is supported.

Also refer to IL_BUSMASTER, page 415.

11.2 Acyclic services (AcyclicCommunication)

11.2.1 IL_SIIISvcRead

Brief description

Function block IL_SIIISvcRead can be used to read parameters of a slave device via the Sercos III service channel.

Target system	Library/folder
MLC	RIL_SercosIII/ AcyclicCommunication
MTX	RIL_SercosIII/ AcyclicCommunication
MLD (IL 2G)	RIL_SercosIII/ AcyclicCommunication

Tab. 11-2: Reference table

Interface description

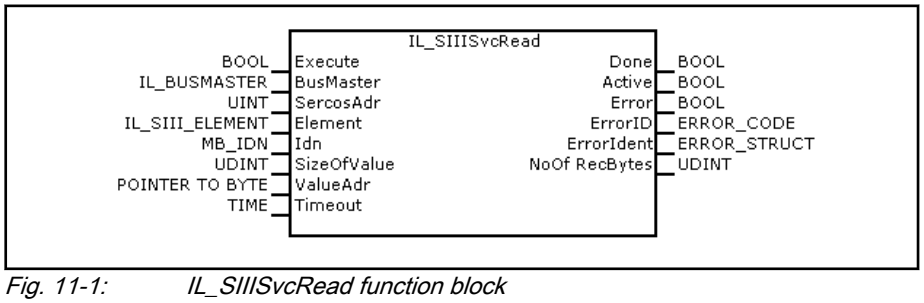


Fig. 11-1: IL_SIIISvcRead function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Element	IL_SIII_ELEMENT	Sercos data block element. This input is used to decide whether name, attribute, unit, minimum value, maximum value or parameter date is read. With Sercos data block element 1 (Element := IL_STATUS;), the service channel "Data Status" is read. Also refer to IL_SIII_ELEMENT, page 414
	Idn	MB_IDN	Parameter IDN. Refer to the function IL_SIIIElementsToIdn, page 408.
	SizeOfValue	UDINT	Size in bytes of buffer or variables provided for data reception (SIZEOF(Value))
	ValueAdr	POINTER TO BYTE	Address of buffer or variables provided for data reception.
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	NoOfRecBytes	UDINT	Number of bytes copied to the "ValueAdr" buffer.

Tab. 11-3: IL_SIIISvcRead interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”
Element	IL_SIII_ELEMENT	1	7	7	Rising edge at “Execute”
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
SizeOfValue	UDINT	0	16#FFFF	0	Rising edge at “Execute”

Name	Type	Min. value	Max. value	Default value	Effective
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-4: Min./max. and default values of the IL_SIIISvcRead inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found either in error table **F_RELATED_TABLE** (ERROR_TABLE = 16#0170) or bzw. **SERCOS_TABLE** (ERROR_TABLE = 16#0010).

ErrorID	Additional1	Additional2	
Errors of the function block in the error table **F_RELATED_TABLE** (ERROR_TABLE = 16#0170)			
INPUT_RANGE_ERROR	16#00000002	16#00000001	The input value of the parameter "BusMaster" is unequal to IL_BUSMASTER_0 and outside of the input value range
INPUT_RANGE_ERROR	16#00000002	16#00000002	The input value of the parameter "SercosAdr" is 0 or >511 MLD: The input value of the parameter "SercosAdr" > 511
INPUT_RANGE_ERROR	16#00000002	16#00000003	The input value of the parameter "Element" is <> 7
INPUT_RANGE_ERROR	16#00000002	16#00000004	The input value of the parameter "Idn" is > 16#FFFFFFFF
INPUT_RANGE_ERROR	16#00000002	16#00000006	The input value of the parameter "ValueAdr" is 0
INPUT_RANGE_ERROR	16#00000002	16#00000007	The input value of the parameter "Timeout" is >T#60 min.
STATE_MACHINE_ERROR	16#00000006	16#00000001	Error in state machine
OTHER_ERROR	16#00000015	16#00000001	A time monitoring error was detected
Errors of the function block in the error table **SERCOS_TABLE** (ERROR_TABLE = 16#0010)			
COMMUNICATION_ERROR	16#00008002	16#00000000	The input value of the parameter "SizeOfValue" is too small. The received data exceeds the memory provided

Tab. 11-5: Error codes of function block IL_SIIISvcRead in F_RELATED_TABLE := 16#0170 or SERCOS_TABLE (ERROR_TABLE = 16#0010).

Examples

Reading single parameters and data function block elements

When reading single parameters and data function block elements, the data type of the variables at the "ValueAdr" input **should** fit to the data size of the element to be read.

In the following example, the Sercos parameter S-0-1002.0.0, Sercos cycle time, is read.

All Sercos data block elements of the parameter are read.

Auxiliary structures

```
TYPE LIST_STRING:
STRUCT
 uiCurLength: UINT;
 uiMaxLength: UINT;
 strValue:    STRING;
END_STRUCT
END_TYPE
```

Variable definition

```
VAR
 fbSIIISvcRead:                  IL_SIIISvcRead;

 uiServiceChannelStatus_E1:      UINT;
 tSercosCycleTimeName_E2:        LIST_STRING;
 dwSercosCycleTime_Attribute_E3: DWORD;
 tSercosCycleTime_Unit_E4:       LIST_STRING;
 udiSercosCycleTime_Min_E5:      UDINT;
 udiSercosCycleTime_Max_E6:      UDINT;
 udiSercosCycleTime_Value_E7:    UDINT;
END_VAR
```

Reading data status of service channel

```
 fbSIIISvcRead.Execute:=     TRUE;
 fbSIIISvcRead.SercosAdr:=   65;
 fbSIIISvcRead.Element:=     IL_STATUS;
 fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1002,0,0);
 fbSIIISvcRead.SizeOfValue:= SIZEOF(uiServiceChannelStatus_E1);
 fbSIIISvcRead.ValueAdr:=    ADR(uiServiceChannelStatus_E1);

 fbSIIISvcRead();
```

Reading parameter name

```
 fbSIIISvcRead.Execute:=     TRUE;
 fbSIIISvcRead.SercosAdr:=   65;
 fbSIIISvcRead.Element:=     IL_NAME;
 fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1002,0,0);
 fbSIIISvcRead.SizeOfValue:= SIZEOF(tSercosCycleTimeName_E2);
 fbSIIISvcRead.ValueAdr:=    ADR(tSercosCycleTimeName_E2);

 fbSIIISvcRead();

IF fbSIIISvcRead.Done = TRUE THEN
 tSercosCycleTimeName_E2.strValue[tSercosCycleTimeName_E2.uiCurLength]:=0;
END_IF
```

Reading parameter attribute

```
 fbSIIISvcRead.Execute:=     TRUE;
 fbSIIISvcRead.SercosAdr:=   65;
 fbSIIISvcRead.Element:=     IL_ATTRIBUTE;
 fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1002,0,0);
 fbSIIISvcRead.SizeOfValue:= SIZEOF(dwSercosCycleTime_Attribute_E3);
 fbSIIISvcRead.ValueAdr:=    ADR(dwSercosCycleTime_Attribute_E3);

 fbSIIISvcRead();
```

Reading parameter unit

```
 fbSIIISvcRead.Execute:=     TRUE;
 fbSIIISvcRead.SercosAdr:=   65;
 fbSIIISvcRead.Element:=     IL_UNIT;
 fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1002,0,0);
 fbSIIISvcRead.SizeOfValue:= SIZEOF(tSercosCycleTime_Unit_E4);
 fbSIIISvcRead.ValueAdr:=    ADR(tSercosCycleTime_Unit_E4);

 fbSIIISvcRead();

IF fbSIIISvcRead.Done = TRUE THEN
 tSercosCycleTimeUnit_E4.strValue[tSercosCycleTimeUnit_E4.uiCurLength]:=0;
END_IF
```

Reading parameter minimum value

```
 fbSIIISvcRead.Execute:=     TRUE;
 fbSIIISvcRead.SercosAdr:=   65;
 fbSIIISvcRead.Element:=     IL_MINVALUE;
 fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1002,0,0);
 fbSIIISvcRead.SizeOfValue:= SIZEOF(udiSercosCycleTime_Min_E5);
 fbSIIISvcRead.ValueAdr:=    ADR(udiSercosCycleTime_Min_E5);

 fbSIIISvcRead();
```

Reading parameter maximum value

```
fbSIIISvcRead.Execute:=     TRUE;
fbSIIISvcRead.SercosAdr:=   65;
fbSIIISvcRead.Element:=     IL_MAXVALUE;
fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1002,0,0);
fbSIIISvcRead.SizeOfValue:= SIZEOF(udiSercosCycleTime_Max_E6);
fbSIIISvcRead.ValueAdr:=    ADR(udiSercosCycleTime_Max_E6);

fbSIIISvcRead();
```

Reading parameter date

```
fbSIIISvcRead.Execute:=     TRUE;
fbSIIISvcRead.SercosAdr:=   65;
fbSIIISvcRead.Element:=     IL_OPDATA;
fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1002,0,0);
fbSIIISvcRead.SizeOfValue:= SIZEOF(udiSercosCycleTime_Value_E7);
fbSIIISvcRead.ValueAdr:=    ADR(udiSercosCycleTime_Value_E7);

fbSIIISvcRead();
```

After successful reading, 100,000 can be found in "ValueAdr" for example.

Parameter S-0-1002 has three decimal places. See attribute.

Thus, the Sercos cycle time is 1,000 µs.

☞ If a parameter with the display format FLOAT is used, the input parameter of "ValueAdr" should be provided with the data type REAL.

For a parameter with a decimal value with decimal places, the data type "Integer" has to be used and the decimal places have to be considered.

Reading lists

In the following example, the Sercos parameter S-0-0017.0.0, IDN list of all operation data, is read.

When reading lists, note that the actual length of a list and the maximum length are part of the list.

2 bytes	2 bytes	<Actual length> bytes
Actual length	Maximum length	User data

Auxiliary structures

```
TYPE LIST_DINT:
 STRUCT
  uiCurLength: UINT;
  uiMaxLength: UINT;
  adiValue:    ARRAY[0..10000] OF DINT;
 END_STRUCT
END_TYPE
```

Variable definition

```
VAR
 fbSIIISvcRead:       IL_SIIISvcRead;
 tListOfIdn_Value_E7: LIST_DINT;
END_VAR
```

Reading parameter date from S-0-0017.0.0 list

```
fbSIIISvcRead.Execute:=     TRUE;
fbSIIISvcRead.SercosAdr:=   65;
```

```
fbSIIISvcRead.Element:=     IL_OPDATA;
fbSIIISvcRead.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,17,0,0);
fbSIIISvcRead.SizeOfValue:= SIZEOF(tListOfIdn_Value_E7);
fbSIIISvcRead.ValueAdr:=    ADR(tListOfIdn_Value_E7);

fbSIIISvcRead();
```

11.2.2 IL_SIIISvcWrite

Brief description

Function block IL_SIIISvcWrite can be used to write parameters of a slave device via the Sercos III service channel.

Target system	Library/folder
MLC	RIL_SercosIII/ AcyclicCommunication
MTX	RIL_SercosIII/ AcyclicCommunication
MLD (IL 2G)	RIL_SercosIII/ AcyclicCommunication

Tab. 11-6: Reference table

Interface description

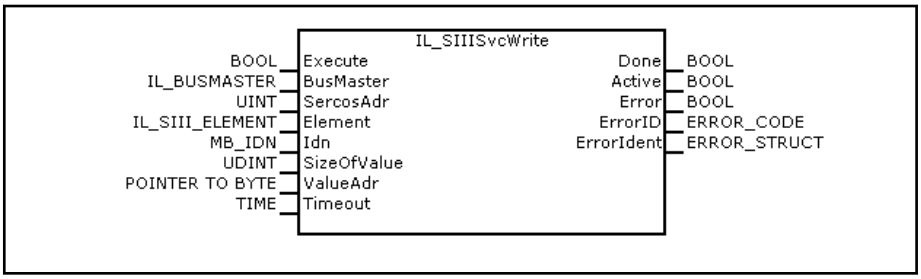


Fig. 11-2: IL_SIIISvcWrite function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Element	IL_SIII_ELEMENT	Sercos data block element. Only IL_OPDATA is currently supported. Also refer to IL_SIII_ELEMENT, page 414
	Idn	MB_IDN	Parameter IDN Refer to the function IL_SIIIElementsToIdn, page 408.
	SizeOfValue	UDINT	Number of bytes to be written via the service channel.
	ValueAdr	POINTER TO BYTE	Address of the buffer or the variables provided for sending data.
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid

I/O type	Name	Data type	Description
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics

Tab. 11-7: IL_SIIISvcWrite interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”
Element	IL_SIII_ELE-MENT	7	7	7	Rising edge at “Execute”
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
SizeOfValue	UDINT	0	16#FFFF	0	Rising edge at “Execute”
ValueAdr	POINTER TO BYTE	N. def.	N. def.	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-8: Min./max. and default values of the IL_SIIISvcWrite inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found in error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#00000002	16#00000001	The input value of the parameter "BusMaster" is unequal to IL_BUSMASTER_0 and outside of the input value range
INPUT_RANGE_ERROR	16#00000002	16#00000002	The input value of the parameter "SercosAdr" is 0 or >511
INPUT_RANGE_ERROR	16#00000002	16#00000003	The input value of the parameter "Element" is <> 7
INPUT_RANGE_ERROR	16#00000002	16#00000004	The input value of the parameter "Idn" is > 16#FFFFFFFF
INPUT_RANGE_ERROR	16#00000002	16#00000005	The input value of the parameter "SizeOfValue" is > 16#FFFF
INPUT_RANGE_ERROR	16#00000002	16#00000006	The input value of the parameter "ValueAdr" is 0
INPUT_RANGE_ERROR	16#00000002	16#00000007	The input value of the parameter "Timeout" is >T#60 min.
RESOURCE_ERROR	16#00000005	16#00000000	The function block is not implemented in the available firmware, e.g. in the MTX Emulation

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#00000006	16#00000001	Error in state machine
OTHER_ERROR	16#00000015	16#00000001	A time monitoring error was detected

Tab. 11-9: Error codes of function block IL_SIIISvcWrite in F_RELATED_TABLE := 16#0170

Examples

Writing single parameters

In the following example, the Sercos parameter S-0-0099.0.0, Reset command, is written.

Variable definition

```
VAR
 uiDummy:    UINT := 3;
 fbSIIISvcWrite: IL_SIIISvcWrite;
END_VAR
```

Writing parameter S-0-0099.0.0

```
 fbSIIISvcWrite.Execute:=     TRUE;
 fbSIIISvcWrite.SercosAdr:=   0;
 fbSIIISvcWrite.Element:=   IL_OPDATA;
 fbSIIISvcWrite.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,99,0,0);
 fbSIIISvcWrite.SizeOfValue:= SIZEOF(uiDummy);
 fbSIIISvcWrite.ValueAdr:=    ADR(uiDummy);

 fbSIIISvcWrite ();
```

Writing lists

In the following example, Sercos parameter S-0-1020.0.0, IP address, is written.

When writing lists, it has to be considered that the actual length of a list and the maximum length are part of the list. Thus, four bytes have to be added to the actual length of the list (see example "fbSIIISvcWrite.SizeOfValue").

2 bytes	2 bytes	<Actual length> bytes
Actual length	Maximum length	User data

Auxiliary structures

```
TYPE LIST_USINT:
 STRUCT
  uiCurLength: UINT;
  uiMaxLength: UINT;
  ausiValue:   ARRAY[0..3] OF USINT;
 END_STRUCT
END_TYPE
```

Variable definition

```
VAR
 fbSIIISvcWrite:          IL_SIIISvcWrite;
 // IndraLogic 2G Syntax
 tIpAddress: LIST_USINT:= (uiCurLength:=4, ausiValue:=[192,186,73,1]);
 (* IndraLogic 1x Syntax
 tIpAddress: LIST_USINT:= (uiCurLength:=4, ausiValue:=192,186,73,1);*)
END_VAR
```

Writing parameter S-0-1020.0.0

```
 fbSIIISvcWrite.Execute:=     TRUE;
 fbSIIISvcWrite.SercosAdr:=   65;
 fbSIIISvcWrite.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,1020,0,0);
```

```
fbSIIISvcWrite.SizeOfValue:= tIpAddress.uiCurLength + 4;
fbSIIISvcWrite.ValueAdr:=    ADR(tIpAddress);

fbSIIISvcWrite ();
```

11.3 Acyclic services/parameters (AcyclicCommunication/Parameter)

11.3.1 IL_SIIICommand

Brief description Function block IL_SIIICommand can be used to execute commands of a slave device via the Sercos III service channel.

Target system	Library/folder
MLC	RIL_SercosIII/AcyclicCommunication/parameter
MTX	RIL_SercosIII/AcyclicCommunication/parameter
MLD (IL 2G)	RIL_SercosIII/AcyclicCommunication/parameter

Tab. 11-10: Reference table of the IL_SIIICommand function block

Interface description

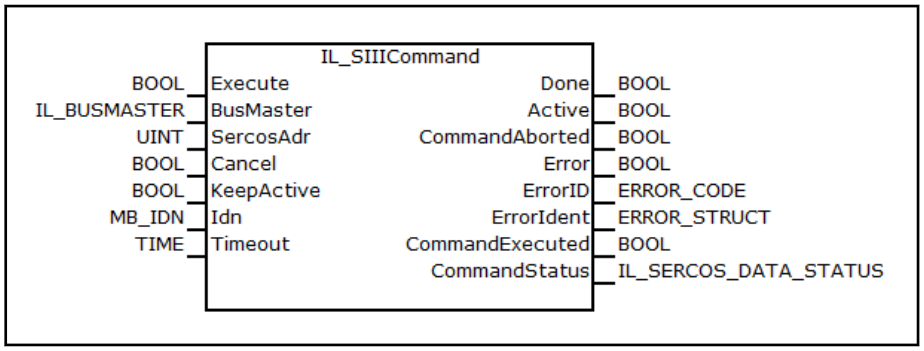


Fig. 11-3: IL_SIIICommand function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Cancel	BOOL	Aborts the running command at a rising edge
	KeepActive	BOOL	Keeps the command active after execution
	Idn	MB_IDN	Parameter IDN Refer to the functions IL_SIIIElementsToIdn, page 408, or IL_SIIIStringToIdn, page 412.
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.

I/O type	Name	Data type	Description
	Active	BOOL	Processing active, output data invalid.
	CommandAborted	BOOL	Command successfully aborted and reset
	Error	BOOL	Processing completed with errors.
	ErrorID	ERROR_ CODE	Diagnostic description in case of error.
	ErrorIdent	ERROR_ STRUCT	Detailed diagnostics
	CommandExecuted	BOOL	Command successfully completed
	CommandStatus	IL_SERCOS_DATA_STATUS	Status of the selected parameter IL_SERCOS_DATA_STATUS, page 361

Tab. 11-11: IL_SIIICommand interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”
Cancel	BOOL			FALSE	Continuous
KeepActive	BOOL			FALSE	Continuous
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-12: Min./max. and default values of the IL_SIIICommand inputs

Signal-time diagram

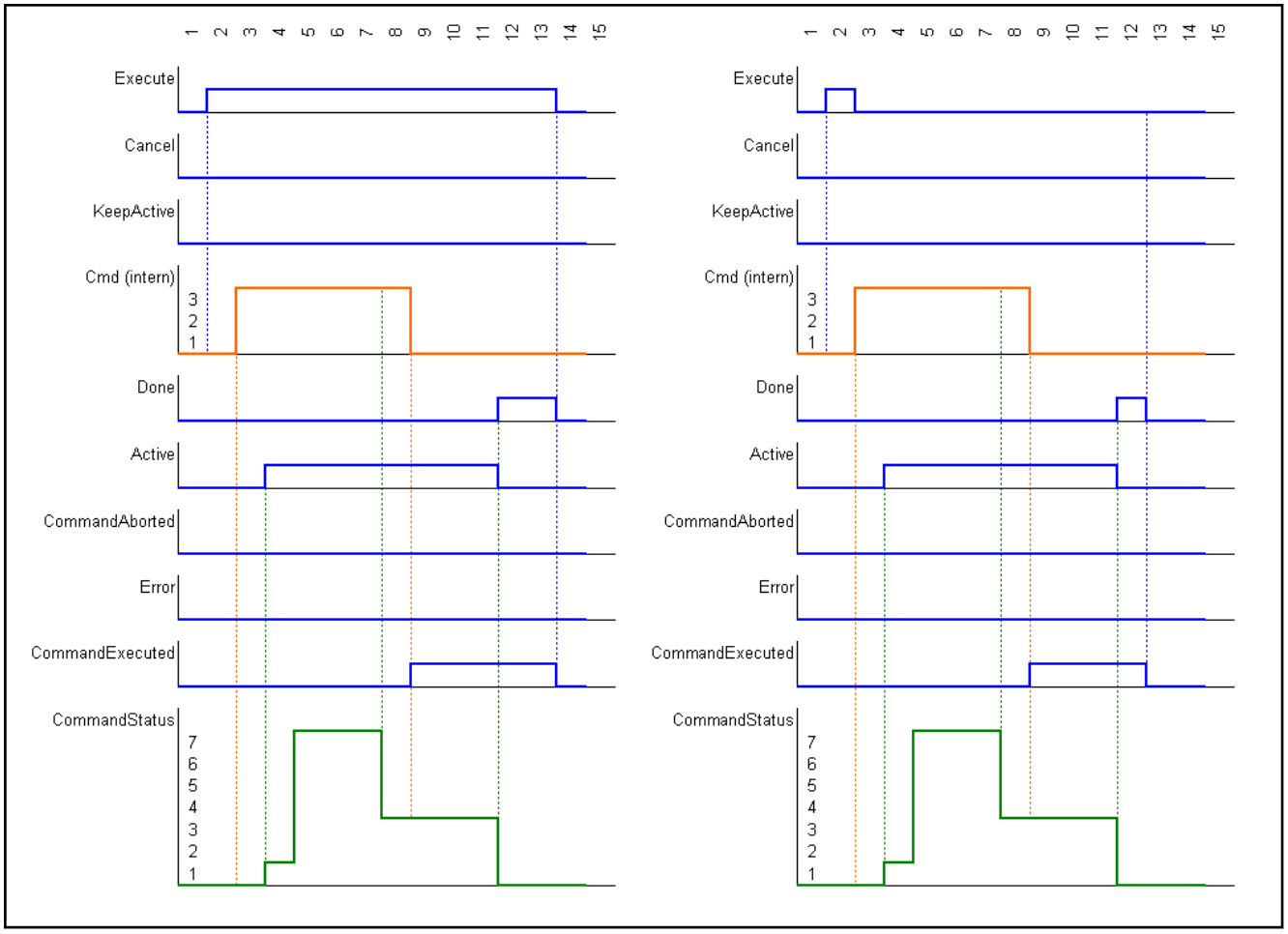


Fig. 11-4: Signal-time diagram for signal flow "Execute-Done"

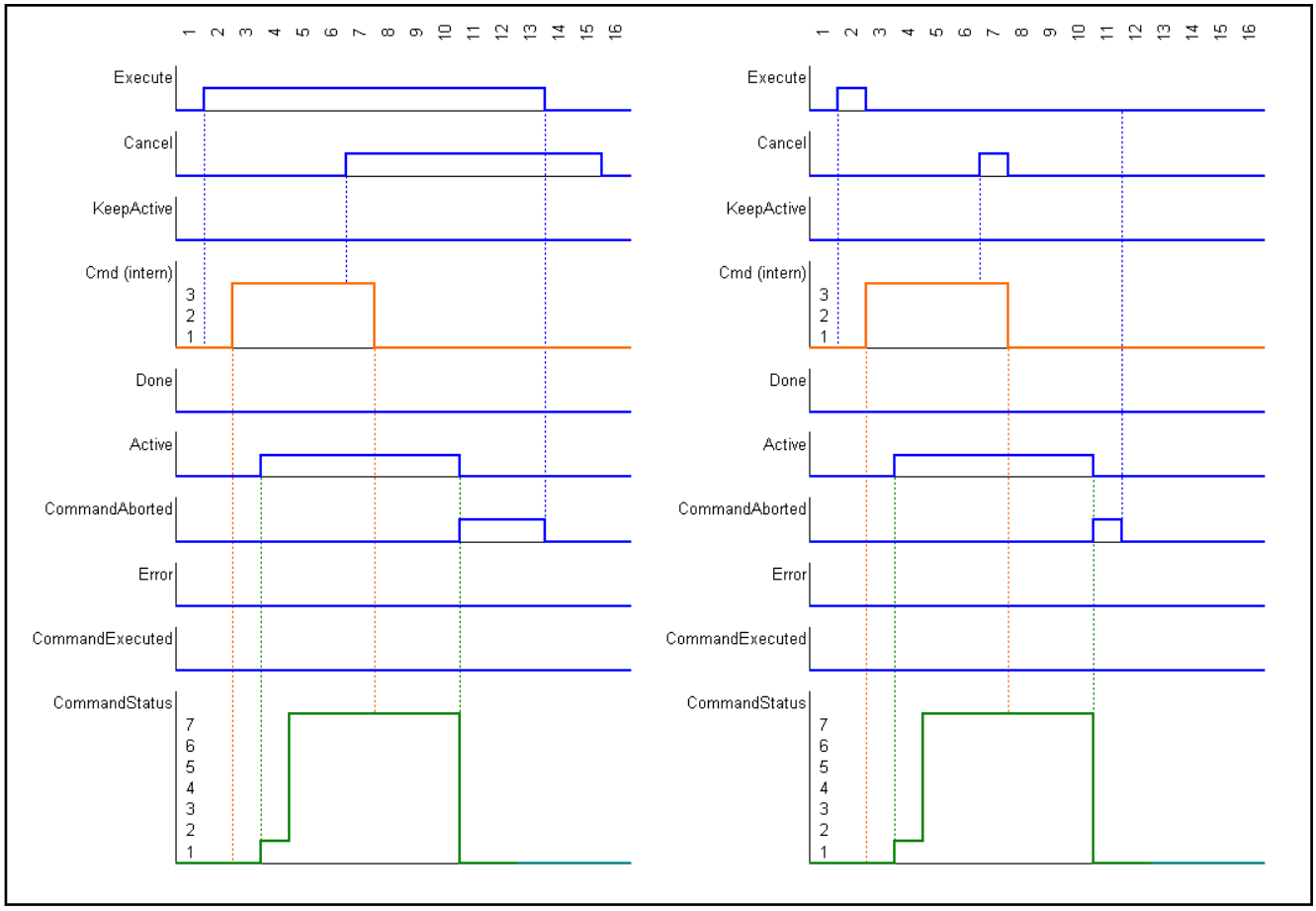


Fig. 11-5: Signal-time diagram for signal flow "Execute-Cancel-CommandAborted"

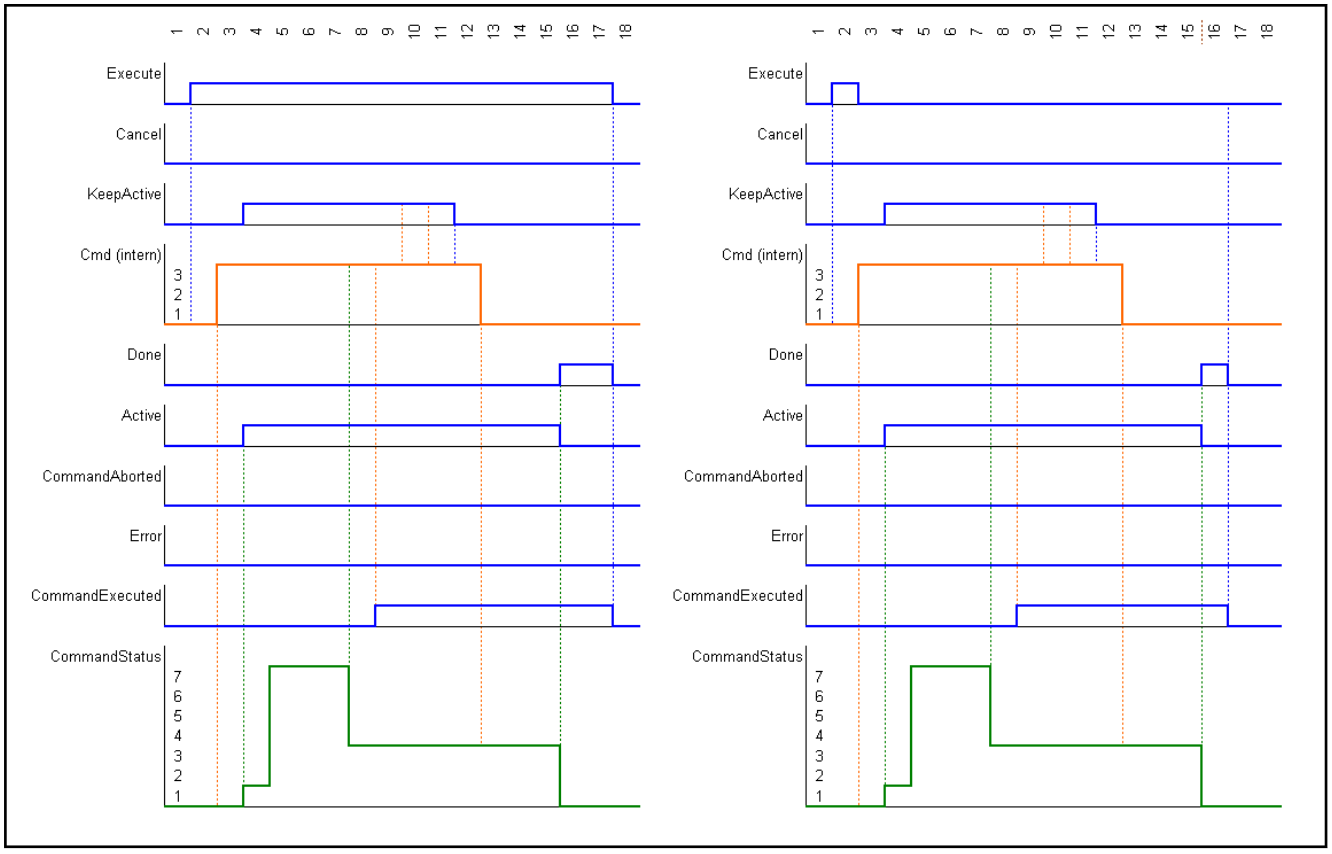


Fig. 11-6: Signal-time diagram for signal flow "Execute-KeepActive-Done"

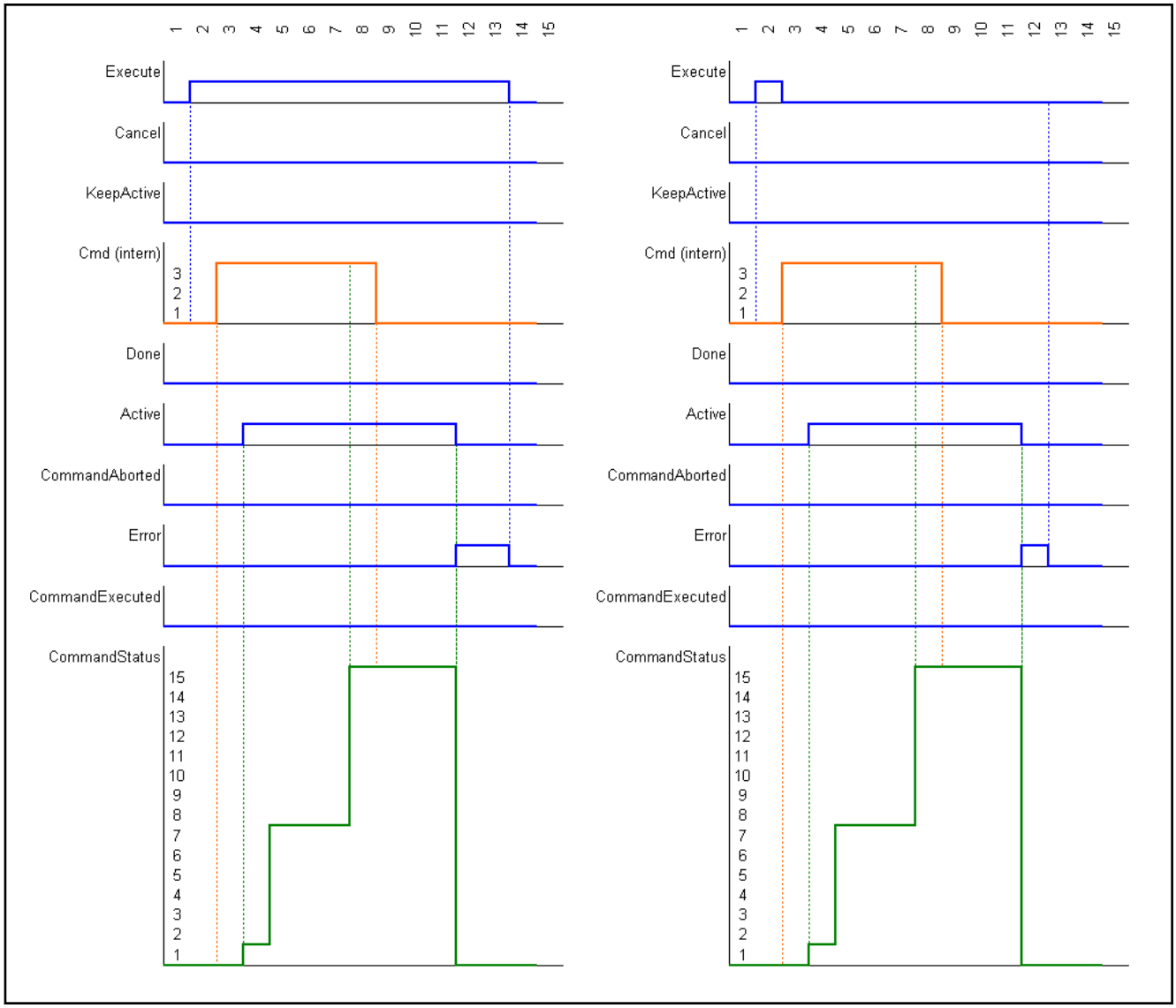


Fig. 11-7: Signal-time diagram for signal flow "Execute-Error"

Description

The function block "IL_SIIICommand" starts the execution of the command defined via the "Idn" input after having enabled processing via "Execute". In general, the command is automatically reset after processing.

Depending on the respective command, its execution can be canceled at any time via the "Cancel" input.

The "KeepActive" input allows the command to remain active after a successful execution. If the input "KeepActive" is deactivated later on, the command is automatically reset.

The successful completion or successful abort of a command is displayed via the outputs "Done" or "CommandAborted" for at least one call.

The active processing of the command is displayed at the "Active" output and the respective date at the "CommandStatus" output.

The current status of the command execution is permanently displayed at the "CommandStatus" output and the completion of the command execution is signaled at the "CommandExecuted" output.

If an error occurs while processing the command, this is signaled by the"Error" output and by "ErrorID", "ErrorIdent" for at least one call.

CommandStatus:

- 0 - Command not set and not released
- 1 - Command set and released
- 3 - Command processed correctly
- 5 - Command processing interrupted
- 7 - Command processing running
- 15 - Error, command processing impossible
- 256 - Value of the parameter beyond its min./max. range

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

The errors of the internally used function blocks IL_SIIISvcRead Troubleshooting, page 335, and IL_SIIISvcWrite Troubleshooting, page 339, as well as the following errors can be found in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002905	Attribute value	Parameter is no command

Tab. 11-13: Error codes of function block IL_SIIICommand in F_RELATED_TABLE := 16#0170

Example on automatic Reset

Executing command S-0-0099

In the following example, the Sercos command S-0-0099.0.0, C0500 Reset state class 1, is executed.

Variable definition

```
VAR
     fbSIIICommand:    IL_SIIICommand;
END_VAR
```

Executing program code S-0-0099

```
fbSIIICommand.Execute:=   TRUE;
fbSIIICommand.SercosAdr:= 1;
fbSIIICommand.Idn:= IL_SIIIElementsToIdn(IL_S_PARAM,0,0099,0,0);

fbSIIICommand();

IF fbSIIICommand.Done THEN
  ; (* command executed successfull *)
END_IF
```

Error is deleted after successful execution (if the error caused was eliminated before).

Example on "Cancel"

Executing command S-0-0139

In the following, the Sercos command S-0-0139.0.0, C1600 Command parking axis is executed for 5 seconds and than stopped using the "Cancel" input.

The command can only be executed in Sercos phase 2 or in the parameterization mode PM.

Variable definition

```
VAR
     fbSIIICommand:    IL_SIIICommand;
```

```
  fbTON:          TON;
END_VAR
```

Executing program code S-0-0139

```
fbSIIICommand.Execute:=    TRUE;
fbSIIICommand.SercosAdr:=  1;
fbSIIICommand.Idn:=        IL_SIIIElementsToIdn(IL_S_PARAM,0,0139,0,0);

IF fbSIIICommand.CommandStatus = IL_SERCOS_DATA_STATUS.IL_SERCOS_DATA_IN_PROCESS THEN
  (* command started successfull, start timer *)
  fbTON(IN:= TRUE, PT:= T#5S );
  (* Cancel the command after 5 seconds *)
  fbSIIICommand.Cancel := fbTON.Q;
END_IF

fbSIIICommand();

IF fbSIIICommand.Done THEN
  ; (* finished *)
END_IF
```

After the successful execution, the command "parking axis" is stopped again.

Example on KeepActive

Executing command S-0-0149

In the following, the Sercos command S-0-0149.0.0, C1300 Command travel to fixed stop is executed and then canceled by resetting the "KeepActive" input.

The command can only be executed using drive enabled.

Variable definition

```
VAR
     fbSIIICommand:    IL_SIIICommand;
  fbTON:          TON;
END_VAR
```

Executing program code S-0-0149

```
fbSIIICommand.Execute:=    TRUE;
fbSIIICommand.SercosAdr:=  1;
fbSIIICommand.Idn:=        IL_SIIIElementsToIdn(IL_S_PARAM,0,0149,0,0);

IF fbSIIICommand.CommandStatus = IL_SERCOS_DATA_STATUS.IL_SERCOS_DATA_END_OK THEN
  (* command started successfull, start timer *)
  fbTON(IN:= TRUE, PT:= T#5S );
END_IF
(* Cancel the command after 5 seconds *)
fbSIIICommand.KeepActive := NOT fbTON.Q;

fbSIIICommand();

IF fbSIIICommand.Done THEN
  ; (* finished *)
END_IF
```

After the successful execution, the command "Move to fixed stop" is stopped again.

11.3.2 IL_SIIIReadParameter

Brief description

The function block IL_SIIIReadParameter can be used to read 2-byte and 4-byte parameters of a slave device via the Sercos III service channel. The parameter value can be converted into a DINT value considering the parameter attribute.

The Value (DINT) output has to be converted to the correct PLC data type with regard to the data type of the parameter. The data lengths of 2 and 4 bytes respectively are not significant if 4-byte data types are always used on the PLC.

- Data type DEC (decimal with sign): "Value" can be copied to a DINT variable
- Data type UDEC (decimal without sign): Converting "Value" with DINT_TO_UDINT to a UDINT variable
- Data types BINARY, HEX and IDN: Converting "Value" with DINT_TO_DWORD to a DWORD variable

For examples on the respective data types, see below.

Also refer to the function IL_SIIIAttribute, page 405, and the data type IL_SIII_DATA_TYPE, page 414.

The IL_SIIIReadParameter function block provides the required information for a correct conversion at the "Attribute" output (for analysis, use the IL_SIIIAttribute function).

Target system	Library/folder
MLC	RIL_SercosIII/AcyclicCommunication/parameter
MTX	RIL_SercosIII/AcyclicCommunication/parameter
MLD (IL 2G)	RIL_SercosIII/AcyclicCommunication/parameter

Tab. 11-14: Reference table

Interface description

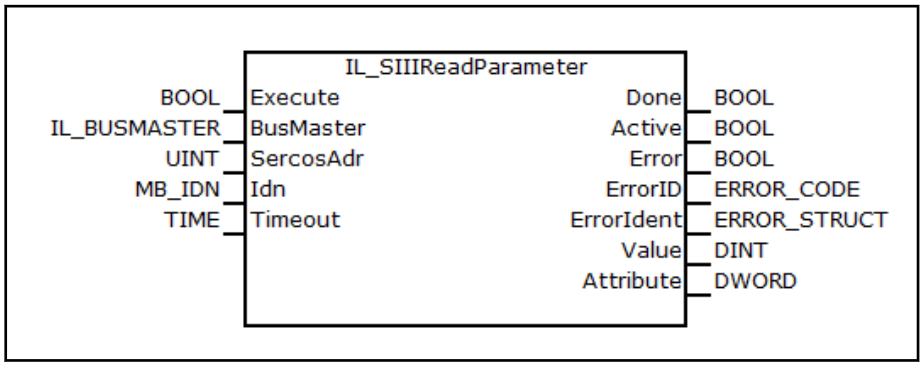


Fig. 11-8: IL_SIIIReadParameter function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUS-MASTER_0. IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Idn	MB_IDN	Parameter IDN. Refer to the functions IL_SIIIElementsToIdn, page 408, or IL_SIIIStringToIdn, page 412.

I/O type	Name	Data type	Description
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with errors.
	ErrorID	ERROR_CODE	Diagnostic description in case of error.
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	Value	DINT	Parameter value
	Attributes	DWORD	Parameter attribute

Tab. 11-15: IL_SIIIReadParameter interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-16: Min./max. and default values of the IL_SIIIReadParameter inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

The errors of the internally used function block IL_SIIISvcRead Troubleshooting, page 335, as well as the following function block errors can be found in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002900	Attribute value	Data length 8-byte parameter or list cannot be converted
INPUT_RANGE_ERROR	16#00002904	Attribute value	Only the parameter types UDEC, DEC, BINARY, HEX or IDN can be converted.

Tab. 11-17: Error codes of function block IL_SIIIReadParameter in F_RELATED_TABLE := 16#0170

Example on the data type DEC (decimal with sign)

Reading S-0-0051

In the following example, the Sercos parameter S-0-0051.0.0, Actual position encoder 1, is read.

Variable definition

```
VAR
        fbSIIIReadParameter:    IL_SIIIReadParameter;
```

```
   diValue: DINT;
END_VAR
```

Reading program code S-0-0051

```
fbSIIIReadParameter.Execute:= TRUE;
fbSIIIReadParameter.SercosAdr:= 1;
fbSIIIReadParameter.Idn:=IL_SIIIElementsToIdn(IL_S_PARAM,0,0051,0,0);

fbSIIIReadParameter();

IF fbSIIIReadParameter.Done = TRUE THEN
  (* S-0-0051.0.0 is signed decimal -> copy direct *)
  diValue := fbSIIIReadParameter.Value;
END_IF
```

After successful reading, "1234567" is found in "diValue" for example.

Example on the data type UDEC (decimal with sign)

Reading S-0-0091

In the following example, the Sercos parameter S-0-0091.0.0, Velocity limit value bipolar, is read.

Variable definition

```
 VAR
     fbSIIIReadParameter:    IL_SIIIReadParameter;
  udiValue: UDINT;
END_VAR
```

Reading program code S-0-0091

```
fbSIIIReadParameter.Execute:= TRUE;
fbSIIIReadParameter.SercosAdr:= 1;
fbSIIIReadParameter.Idn:=IL_SIIIElementsToIdn(IL_S_PARAM,0,0091,0,0);

fbSIIIReadParameter();

IF fbSIIIReadParameter.Done = TRUE THEN
  (* S-0-0091.0.0 is unsigned decimal -> convert *)
  udiValue := DINT_TO_UDINT(fbSIIIReadParameter.Value);
END_IF
```

After successful reading, "10000000" is found in "udiValue" for example.

Example on the binary data type (also applies to HEX and IDN)

Reading S-0-0135

In the following example, the Sercos parameter S-0-0135.0.0, Drive status, is read.

Variable definition

```
 VAR
     fbSIIIReadParameter:    IL_SIIIReadParameter;
  wValue: WORD;
END_VAR
```

Reading program code S-0-0135

```
fbSIIIReadParameter.Execute:=   TRUE;
fbSIIIReadParameter.SercosAdr:= 1;
fbSIIIReadParameter.Idn:= IL_SIIIElementsToIdn(IL_S_PARAM,0,0135,0,0);

fbSIIIReadParameter();

IF fbSIIIReadParameter.Done = TRUE THEN
  (* S-0-0135.0.0 is binary -> convert *)
  wValue := DINT_TO_WORD(fbSIIIReadParameter.Value);
END_IF
```

After successful reading, "16#4000" is found in "wValue" for example.

Example on reading including data type analysis

Reading S-0-0390 with attribute analysis

In the following example, the Sercos parameter S-0-0390.0.0, Diagnostic number, is read. The attribute is not required in this case as the S-0-0390 data type is known. The example shows a possible analysis when reading parameters with different attributes.

Variable definition

```
VAR
     fbSIIIReadParameter:    IL_SIIIReadParameter;
  diValue: DINT;
  iValue: INT;
  udiValue: UDINT;
  uiValue: UINT;
  dwValue: DWORD;
  wValue: WORD;
  enDataType:     IL_SIII_DATA_TYPE;
  enDataLength:   IL_SIII_DATA_LENGTH;
END_VAR
```

Reading program code S-0-0390 with attribute analysis

```
fbSIIIReadParameter.Execute:=   TRUE;
fbSIIIReadParameter.SercosAdr:= 1;
fbSIIIReadParameter.Idn:= IL_SIIIElementsToIdn(IL_S_PARAM,0,0390,0,0);

fbSIIIReadParameter();

IF fbSIIIReadParameter.Done = TRUE THEN
  IL_SIIIAttribute(Attribute:= fbSIIIReadParameter.Attribute ,
                   DecPoints=> ,
                   DataType=>enDataType ,
                   DataLength=>enDataLength );
  CASE enDataType OF
    IL_SIII_DATA_TYPE.DEC:
      (* signed decimal *)
      IF enDataLength = IL_SIII_DATA_LENGTH.FIX_4_BYTE THEN
        diValue := fbSIIIReadParameter.Value;
      ELSE  (* 2 byte *)
        iValue:= DINT_TO_INT(fbSIIIReadParameter.Value);
      END_IF

    IL_SIII_DATA_TYPE.UDEC:
      (* unsigned decimal *)
      IF enDataLength = IL_SIII_DATA_LENGTH.FIX_4_BYTE THEN
        udiValue:= DINT_TO_UDINT(fbSIIIReadParameter.Value);
      ELSE (* 2 byte *)
        uiValue:= DINT_TO_UINT(fbSIIIReadParameter.Value);
      END_IF

    IL_SIII_DATA_TYPE.BINARY,
    IL_SIII_DATA_TYPE.HEX,
    IL_SIII_DATA_TYPE.IDN :
      (* all in (D)WORD *)
      IF enDataLength = IL_SIII_DATA_LENGTH.FIX_4_BYTE THEN
        dwValue := DINT_TO_DWORD(fbSIIIReadParameter.Value);
      ELSE (* 2 byte *)
        wValue := DINT_TO_WORD(fbSIIIReadParameter.Value);
      END_IF

    ELSE
    ; (* not possible (would be fbSIIIReadParameter.Error) *)
    END_CASE
END_IF
```

After successful reading, "16#000A0013" is found in "dwValue" for example.

11.3.3 IL_SIIIReadRealParameter

Brief description The function block IL_SIIIReadRealParameter can be used to read 2-byte and 4-byte parameters of a slave device via the Sercos III service channel. The parameter value can be converted into a REAL value considering the attribute.

Target system	Library/folder
MLC	RIL_SercosIII/AcyclicCommunication/parameter
MTX	RIL_SercosIII/AcyclicCommunication/parameter
MLD (IL 2G)	RIL_SercosIII/AcyclicCommunication/parameter

Tab. 11-18: Reference table

Interface description

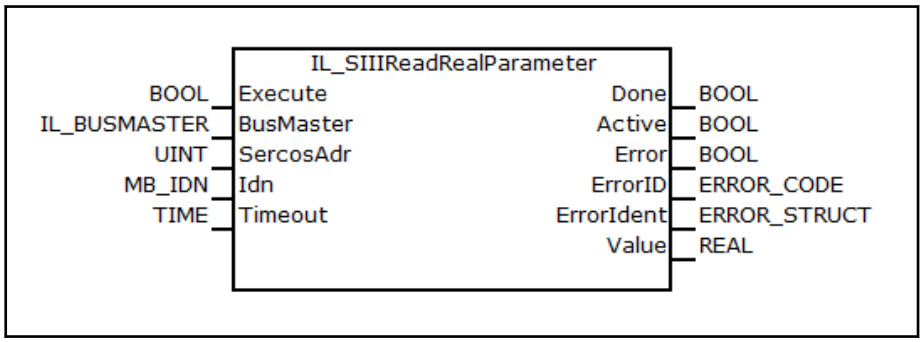


Fig. 11-9: IL_SIIIReadRealParameter function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0 IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device
	Idn	MB_IDN	Parameter IDN. Refer to the functions IL_SIIIElementsToIdn, page 408, or IL_SIIIStringToIdn, page 412.
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	Value	REAL	Parameter value

Tab. 11-19: IL_SIIIReadRealParameterinterface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”

Name	Type	Min. value	Max. value	Default value	Effective
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-20: Min./max. and default values of the IL_SIIIReadRealParameter inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

The errors of the internally used function block IL_SIIISvcRead Troubleshooting, page 335, as well as the following function block errors can be found in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002900	Attribute value	Data length 8-byte parameter or list cannot be converted
INPUT_RANGE_ERROR	16#00002901	Attribute value	Only the parameter types UDEC, DEC or FLOAT can be converted.

Tab. 11-21: Error codes of function block IL_SIIIReadRealParameter in F_RELATED_TABLE := 16#0170

Example

Reading S-0-0051

In the following example, the Sercos parameter S-0-0051.0.0, Actual position encoder 1, is read.

Variable definition

```
 VAR
     fbSIIIReadRealParameter:    IL_SIIIReadRealParameter;
     rValue:    REAL;
END_VAR
```

Reading program code S-0-0051

```
fbSIIIReadRealParameter.Execute:=     TRUE;
fbSIIIReadRealParameter.SercosAdr:=   1;
fbSIIIReadRealParameter.Idn:=   IL_SIIIElementsToIdn(IL_S_PARAM,0,0051,0,0);

fbSIIIReadRealParameter();

IF fbSIIIReadRealParameter.Done = TRUE THEN
  rValue := fbSIIIReadRealParameter.Value;
END_IF
```

After successful reading, "123.4567" is written in "rValue" for example.

11.3.4 IL_SIIIReadStringParameter

Brief description

The function block IL_SIIIReadStringParameter can be used to read text lists of a slave device via the Sercos III service channel.

Target system	Library/folder
MLC	RIL_SercosIII/AcyclicCommunication/parameter
MTX	RIL_SercosIII/AcyclicCommunication/parameter
MLD (IL 2G)	RIL_SercosIII/AcyclicCommunication/parameter

Tab. 11-22: Reference table

Interface description

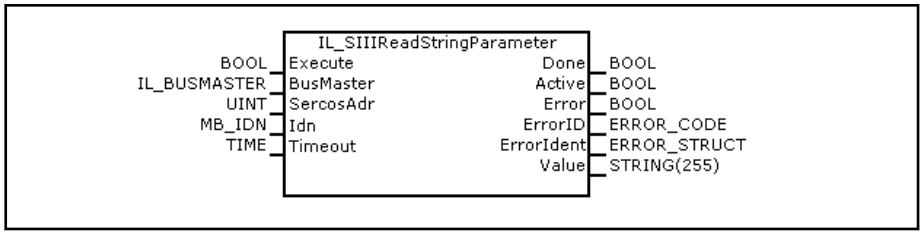


Fig. 11-10: IL_SIIIReadStringParameter function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Idn	MB_IDN	Parameter IDN Refer to the functions IL_SIIIElementsToIdn, page 408, or IL_SIIIStringToIdn, page 412.
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with errors.
	ErrorID	ERROR_CODE	Diagnostic description in case of error.
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	Value	STRING(255)	Character string with up to 255 characters

Tab. 11-23: IL_SIIIReadStringParameter interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-24: Min./max. and default values of the IL_SIIIReadStringParameter inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

The errors of the internally used function block IL_SIIISvcRead Troubleshooting, page 335, as well as the following function block errors can be found in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002902	Attribute value	Character string in parameter too long (max. 254 bytes + \0)
INPUT_RANGE_ERROR	16#00002903	Attribute value	Only parameters of the data type “Text” and length of one byte of variable list can be read

Tab. 11-25: Error codes of function block IL_SIIIReadStringParameter in F_RELATED_TABLE := 16#0170

Example **Reading S-0-0095**

In the following example, the Sercos parameter S-0-0095.0.0, Diagnostics, is read.

Variable definition

```
 VAR
     fbSIIIReadStringParameter:    IL_SIIIReadStringParameter;
     strValue:    STRING;
END_VAR
```

Reading program code S-0-0095

```
fbSIIIReadStringParameter.Execute:=     TRUE;
fbSIIIReadStringParameter.SercosAdr:=   1;
fbSIIIReadStringParameter.Idn:=  IL_SIIIElementsToIdn(IL_S_PARAM,0,0095,0,0);

fbSIIIReadStringParameter();

IF fbSIIIReadStringParameter.Done = TRUE THEN
  strValue := fbSIIIReadStringParameter.Value;
END_IF
```

After successful reading, the value "A0013 Ready for power on" can for example be found in "strValue".

11.3.5 IL_SIIIWriteParameter

Brief description The function block IL_SIIIWriteParameter can be used to write 2-byte and 4-byte parameters of a slave device via the Sercos III service channel.



The "Value" (DINT) input has to be converted according to the data type of the parameter.

The data lengths of 2 and 4 bytes respectively are not significant if 4-byte data types are always used on the PLC.

- Data type DEC (decimal with sign): DINT variable can be copied to "Value"
- Data type UDEC (decimal without sign): Convert UDINT variable with UDINT_TO_DINT and copy to "Value"
- Data types BINARY, HEX and IDN: Convert DWORD variable with DWORD_TO_DINT and copy to "Value"

For examples on the respective data types, see below.

Also refer to the function IL_SIIIAttribute, page 405, and the data type IL_SIII_DATA_TYPE, page 414.

Target system	Library/folder
MLC	RIL_SercosIII/AcyclicCommunication/parameter
MTX	RIL_SercosIII/AcyclicCommunication/parameter
MLD (IL 2G)	RIL_SercosIII/AcyclicCommunication/parameter

Tab. 11-26: Reference table

Interface description

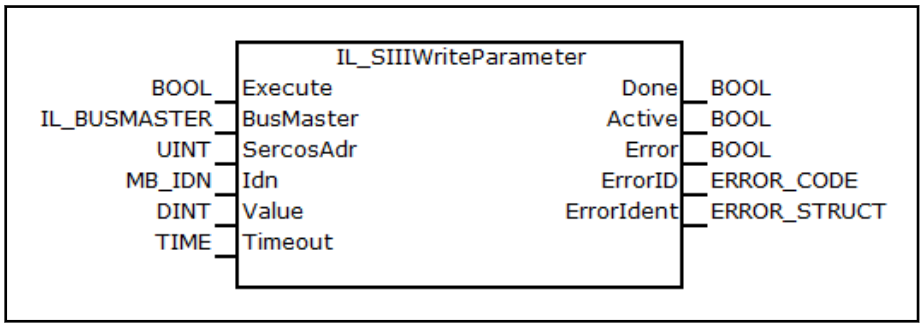


Fig. 11-11: IL_SIIIWriteParameter function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Idn	MB_IDN	Parameter IDN Refer to the functions IL_SIIIElementsToIdn, page 408, or IL_SIIIStringToIdn, page 412.
	Value	DINT	Value to be written to the parameter
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with errors.
	ErrorID	ERROR_CODE	Diagnostic description in case of error.
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics

Tab. 11-27: IL_SIIIWriteParameter interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”

Name	Type	Min. value	Max. value	Default value	Effective
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
Value	DINT	Depending on the data type of the parameter	Depending on the data type of the parameter	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-28: Min./max. and default values of the IL_SIIIWriteParameter inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

The errors of the internally used function block IL_SIIISvcRead Troubleshooting, page 335, as well as the following function block errors can be found in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002900	Attribute value	Data length 8-byte parameter or list cannot be converted
INPUT_RANGE_ERROR	16#00002904	Attribute value	Only the parameter types UDEC, DEC, HEX, BINARY or IDN can be converted.

Tab. 11-29: Error codes of function block IL_SIIIReadRealParameter in F_RELATED_TABLE := 16#0170

Example on the data type DEC (decimal with sign)

Writing S-0-0258

In the following example, the value -42.1234 is written to the Sercos parameter S-0-0258.0.0, Target position.

Variable definition

```
VAR
    fbSIIIWriteParameter:    IL_SIIIWriteParameter;
 diValue: DINT;
END_VAR
```

Writing program code S-0-0258

```
fbSIIIWriteParameter.Execute:=   TRUE;
fbSIIIWriteParameter.SercosAdr:= 1;
fbSIIIWriteParameter.Idn:=       IL_SIIIElementsToIdn(IL_S_PARAM,0,0258,0,0);
diValue := REAL_TO_DINT(-42.1234*10000); (* *10000 because of 4 decimal places
                                            (rotatoric, prefered scaling)
                              S-0-0051.0.0 is signed decimal -> copy direct *)
fbSIIIWriteParameter.Value := diValue;

fbSIIIWriteParameter();

IF fbSIIIWriteParameter.Done = TRUE THEN
  ; (* .... *)
END_IF
```

After a successful writing, "-42.1234" can for example be found in the parameter S-0-0258.

Example on the data type UDEC (decimal with sign)

Writing S-0-0091

In the following example, the value 1042.5678 is written to the Sercos parameter S-0-0091.0.0, velocity limit value.

Variable definition

```
 VAR
     fbSIIIWriteParameter:    IL_SIIIWriteParameter;
  udiValue: UDINT;
END_VAR
```

Writing program code S-0-0091

```
fbSIIIWriteParameter.Execute:=   TRUE;
fbSIIIWriteParameter.SercosAdr:= 1;
fbSIIIWriteParameter.Idn:=       IL_SIIIElementsToIdn(IL_S_PARAM,0,0091,0,0);
udiValue := REAL_TO_UDINT(1042.5678*10000); (* 10000 because of 4 decimal places
                                    (rotatoric, prefered scaling)
               S-0-0091.0.0 is unsigned decimal -> convert from UDINT to DINT) *)
fbSIIIWriteParameter.Value := UDINT_TO_DINT(udiValue);

fbSIIIWriteParameter();

IF fbSIIIWriteParameter.Done = TRUE THEN
  ; (* .... *)
END_IF
```

After a successful writing, "1042.5678" can for example be found in the parameter S-0-0091.

Example on the binary data type (also applies to HEX and IDN)

Writing S-0-0169

In the following example, the value 0b0000.0001.0110.0011 is written to the Sercos parameter S-0-0169.0.0, Touch probe control parameter.

Variable definition

```
 VAR
     fbSIIIWriteParameter:    IL_SIIIWriteParameter;
  wValue: WORD;
END_VAR
```

Reading program code S-0-0169

```
fbSIIIWriteParameter.Execute:=   TRUE;
fbSIIIWriteParameter.SercosAdr:= 1;
fbSIIIWriteParameter.Idn:=       IL_SIIIElementsToIdn(IL_S_PARAM,0,0169,0,0);
wValue := 2#0000000101100011;
               (* S-0-0169.0.0 is 2 byte BINARY -> convert from WORD to DINT *)
fbSIIIWriteParameter.Value := WORD_TO_DINT(wValue);

fbSIIIWriteParameter();

IF fbSIIIWriteParameter.Done = TRUE THEN
  ; (* .... *)
END_IF
```

After a successful writing, "0b0000.0001.0110.0011" can for example be found in the parameter S-0-0169.

11.3.6 IL_SIIIWriteRealParameter

Brief description

The function block IL_SIIIWriteRealParameter can be used to write 2-byte and 4-byte parameters of a slave device via the Sercos III service channel. The value is converted into a REAL value considering the attribute and written to a parameter.

Target system	Library/folder
MLC	RIL_SercosIII/AcyclicCommunication/parameter
MTX	RIL_SercosIII/AcyclicCommunication/parameter
MLD (IL 2G)	RIL_SercosIII/AcyclicCommunication/parameter

Tab. 11-30: Reference table

Interface description

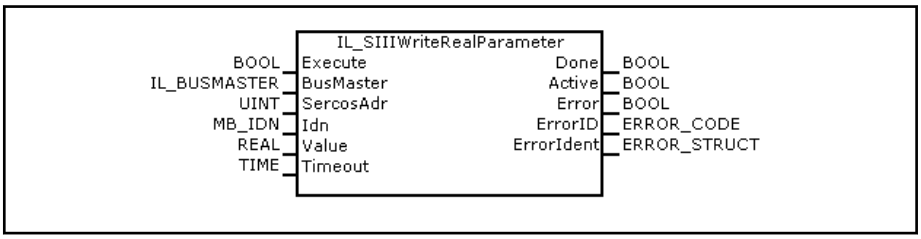


Fig. 11-12: IL_SIIIWriteRealParameter function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Idn	MB_IDN	Parameter IDN. Refer to the functions IL_SIIIElementsToIdn, page 408, or IL_SIIIStringToIdn, page 412.
	Value	REAL	Value to be written to the parameter
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with errors.
	ErrorID	ERROR_CODE	Diagnostic description in case of error.
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics

Tab. 11-31: IL_SIIIWriteRealParameter interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511	0	Rising edge at “Execute”

Name	Type	Min. value	Max. value	Default value	Effective
Idn	MB_IDN	0	16#FFFFFFFF	0	Rising edge at “Execute”
Value	REAL	Depending on data type and decimal places of the parameter	Depending on data type and decimal places of the parameter	0	Rising edge at “Execute”
Timeout	TIME	T#0s	T#60 min	T#2s	Rising edge at “Execute”

Tab. 11-32: Min./max. and default values of the IL_SIIIWriteRealParameter inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

The errors of the internally used function blocks IL_SIIISvcRead Troubleshooting, page 335, and IL_SIIISvcWrite Troubleshooting, page 339, as well as the following errors can be found in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002900	Attribute value	Data length 8-byte parameter or list cannot be converted
INPUT_RANGE_ERROR	16#00002901	Attribute value	Only the parameter types UDEC, DEC or FLOAT can be converted.

Tab. 11-33: Error codes of function block IL_SIIIWriteRealParameter in F_RELATED_TABLE := 16#0170

Example

Writing S-0-0038

In the following example, the value 123.456 is written to the Sercos parameter S-0-0038.0.0, velocity limit value.

Variable definition

```
VAR
    fbSIIIWriteRealParameter:    IL_SIIIWriteRealParameter;
END_VAR
```

Writing program code S-0-0038

```
fbSIIIWriteRealParameter.Execute:=     TRUE;
fbSIIIWriteRealParameter.SercosAdr:=   1;
fbSIIIWriteRealParameter.Idn:=         IL_SIIIElementsToIdn(IL_S_PARAM,0,0038,0,0);
fbSIIIWriteRealParameter.Value:=       123.456;

fbSIIIWriteRealParameter();

IF fbSIIIWriteRealParameter.Done THEN
  ; // finished write successfull
END_IF
```

After a successful writing, "123.4567" can for example be found in the parameter S-0-0038.

11.3.7 IL_SERCOS_DATA_STATUS

Brief description

The status of a command started using the function block IL_SIIICommand, RIL_SercosIII.library, is shown in plain text via the ENUM type "IL_SERCOS_DATA_STATUS".

Target system	Library/folder
MLC	RIL_SercosIII / AcyclicCommunication / Parameter
MTX	RIL_SercosIII / AcyclicCommunication / Parameter
MLC (IL 2G)	RIL_SercosIII / AcyclicCommunication / Parameter

Tab. 11-34: Library assignment

Program:

```
TYPE IL_SERCOS_DATA_STATUS:
(* Status of the parameter *)
(
    IL_SERCOS_DATA_NOT_SET      := 0,     (* Not set and not enabled *)
    IL_SERCOS_DATA_SETTING      := 1,     (* Set and enabled *)
    IL_SERCOS_DATA_END_OK       := 3,     (* Command correctly executed *)
    IL_SERCOS_DATA_INTERRUPTED  := 5,     (* Command execution interrupted *)
    IL_SERCOS_DATA_IN_PROCESS   := 7,     (* In process *)
    IL_SERCOS_DATA_BREAK_ERROR  := 15,    (* Error, command execution impossible*)
    IL_SERCOS_DATA_DATUM_FAULT  := 256    (* Data invalid *)
);
END_TYPE
```

☞ The data status "IL_SERCOS_DATA_DATUM_FAULT" indicates that the value of the parameter id beyond its validity limits, i.e. it is smaller than its minimum value or higher than its maximum value allowed.

11.4 Backing up and restoring (BackupRestore)

11.4.1 IL_SIIIParameterBackup

Brief description

The function block **IL_SIIIParameterBackup** backs up the parameters of a Sercos device in a text-based file on a memory medium of the Sercos master.

The function block is provided with a similar interface as the function blocks ML_AxisBackup and ML_AxisBackup_Type02, but it is only based on the communication mechanisms of the Sercos field bus. This can backup S-parameters and P-parameters, but any number of bus devices, such as I/O devices.

Also refer to

- ML_AxisBackup
- ML_AxisBackup_Type02

Assignment: Target system/library

Target system	Library/folder
MLC	RIL_SercosIII / BackupRestore
MTX	RIL_SercosIII / BackupRestore
MLD (IL 2G)	RIL_SercosIII / BackupRestore

Tab. 11-35: Reference table of the IL_SIIIParameterBackup

Interface description

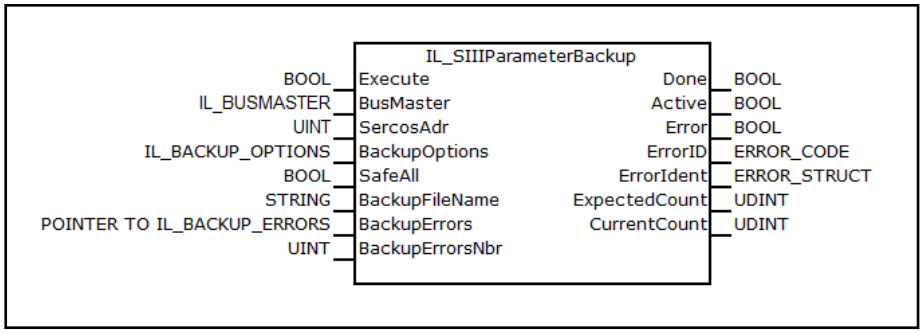


Fig. 11-13: IL_SIIIParameterBackup function block

	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Processing of the function block enabled (edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos address of the slave device. For an MLD, the value 0 has to be used as Sercos address for the local axis.
	BackupOptions	IL_BACKUP_OPTIONS	Scope of the parameter backup IL_BACKUP_OPTIONS, page 374
	SafeAll	BOOL	TRUE: All Sercos elements (IDN, attributes, name, unit, min. and max. values and data) are backed up to a parameter file. FALSE: Only the Sercos elements (IDN, attributes and data) required for backup are backed up
	BackupFileName	STRING	File name used to save the parameter file. Specify the complete path for file systems with different paths.
	BackupErrors	POINTER TO IL_BACKUP_ERRORS	Pointer to an array of IL_BACKUP_ERRORS structures chapter 11.4.3 "IL_BACKUP_ERRORS" on page 374
	BackupErrorsNbr	UINT	Number of arrays in the BackupErrors array
VAR_OUTPUT	Done	BOOL	Processing completed, output data valid
	Active	BOOL	Processing running, output data valid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Error diagnostics (refer to Error codes, page 367)
	ExpectedCount	UDINT	Number of parameters expected for export
	CurrentCount	UDINT	Number of already exported parameters

Tab. 11-36: Interface variables of the IL_SIIIParameterBackup 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	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	511 (MLD) 253 (other systems)	0	Rising edge at “Execute”
BackupOptions	IL_BACK-UP_OPTIONS	ALL_PARAM	TO_SAVE_PAR-AM	TO_SAVE_PARAM	Rising edge at “Execute”
SafeAll	BOOL	FALSE	TRUE	FALSE	Rising edge at “Execute”
BackupFile-Name	STRING	At least one letter	Up to 80 letters including path specification	-	Rising edge at “Execute”
BackupErrors	POINTER TO IL_BACK-UP_ERRORS	> 0			Continuous
BackupEr-rorsNbr	UINT	0	65535	0	Continuous

Tab. 11-37: Min./max. and default values of the function block IL_SIIIParameter-Backup

Functional description

The function block IL_SIIIParameterBackup backs up the S-parameters and the P-parameters of a Sercos device from a text-based file. If the file already exists, it is overwritten. The file can be read again by the IL_SIIIParameter-Restore function block.

BusMaster

This is the Sercos master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0.

SercosAdr

Specify the Sercos address of the device, whose parameters are to be backed up. For an MLD, the address 0 has to be specified for the local axis. Otherwise, the smallest possible Sercos address is MB_MIN_SERCOS_ADDRESS and the highest possible address is MB_MAX_SERCOS_ADDRESS. Depending on the target system , these definitions are provided with different values.

BackupOptions

The parameters to be backed up are selected via the "BackupOptions" input. The following selections are possible:

- ALL_PARAM: All parameters are backed up (content of the list S-0-0017). This option is primarily required to check the configuration
- MODIFIED_PARAM: The parameters deviating from the default values are backed up. This variant is not yet implemented. Thus, S-0-0192 is used.
- TO_SAVE_PARAM: The parameters required for restoration are backed up (S-0-0192). This is the presetting

If the respective Sercos parameter is not available in the Sercos device, no error is generated. Only a parameter file is created without any parameters except for the headers.

SafeAll

If this input is FALSE, only the required data (IDN, attributes, data) is backed up to a parameter file. If the input is TRUE, all data (additionally name, unit, min. and max. values) is specified in a parameter file. Additional data does not have to be backed up when backing up data. It is only useful to back up additional data for a better readability, debugging or determination of valid input values.

Backing up additional files requires more time, as more data has to be read.

BackupFileName

Specify the file name on the memory medium via "BackupFileName". Observe the case sensitivity. Always specify the complete path. The path is not created automatically, e.g. if a path is specified with subdirectories, these subdirectories have to exist. The file does not have to exist. It is either created or overwritten if it already exists.

BackupErrors

This input has to be assigned with a pointer on an IL_BACKUP_ERRORS array. The structure consists of a parameter number (MB_IDN) and of an error code (DWORD). This array lists all errors occurring during the parameter file import. It has to be distinguished between errors occurring in the function block (incorrectly input values applied) and errors resulting while writing parameters. On the latter ones mentioned are entered into this structure. The entered error numbers originate from the function blocks IL_SIIISvcWrite and IL_SIIISvcRead from the SERCOS_TABLE.

At the error output of the function block, a group error is entered in case of one entry in the structure (Additional1 = 16#1E03). This is caused, as reading a parameter is not canceled when an error results. The next parameter is read instead.

BackupErrorsNbr

The number of structures in the "BackupErrors" array is specified. It is recommended to use the "sizeof" operator.

ExpectedCount

This output indicates the total number of parameters to be written. The output is set to 0 in the initialization phase of the function block and is set to its final value while processing. That is, as long as the value is 0, no value has yet been determined.

CurrentCount

This output indicates the number of already read parameters. The output is set to 0 in the initialization phase of the function block and is updated while processing the function block. The value can be used to implement a progress display.

Time response

The function block IL_SIIIParameterBackup works asynchronously. Due to a rising edge at the "Execute" input after the start, it reads the inputs "BusMaster", "SercosAdr", "BackupOptions", "SafeAll" and "BackupFileName" once and the inputs "BackupErrors" and "BackupErrorsNbr" continuously. "Active" is set as long as the function block is operating. After the successful operation, the "Done" output is set. If an error occurs, the "Error" output is set.



Do not use the function block in the Motion task, as it sometimes requires longer execution times.

Compatibility with other parameter files

The function block can create the files required by the IL_SIIIParameterRestore function block. There is also additional information to the parameters in the parameter file that can be different depending on the backup method. The main difference is that the Sercos address is addressed for this function block, but when backing up with IndraWorks, the logical address is addressed instead.

The following table provides an overview on the different backup variants and special features to be considered.

Compatibility between the individual backup/restore function blocks

Backup with	Restore with	Remarks
IL_SercosIIIParameterBackup IndraWorksDS	IL_SercosIIIParameterRestore	✓ Parameter file can be used directly. Both function blocks use the Sercos address as addressing type.
	Only **MLC**: ML_AxisRestore	(✓) Works only if the Sercos address is equal to the logical address.
	Only **MLC**: ML_AxisRestore_Type02	Import does not work. An error is generated (Additional1 = 0x00001E16, Additional2 = 0x000A0008)
IndraWorks	IL_SercosIIIParameterRestore	(✓) When importing, the logical address has to be specified for the "SercosAdrSource" input and the Sercos address for "SercosAdrDestination".
	Only **MLC**: ML_AxisRestore	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
	Only **MLC**: ML_AxisRestore_Type02	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
Only **MLC**: ML_AxisBackup	IL_SercosIIIParameterRestore	Import does not work. An error is generated (Additional1 = 0x00001E03, Additional2 = 0x0000000). Errors with different numbers are entered into the error structure.
	Only **MLC**: ML_AxisRestore	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
	Only **MLC**: ML_AxisRestore_Type02	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
Only **MLC**: ML_AxisRestore_Type02	IL_SercosIIIParameterRestore	(✓) When importing, the logical address has to be specified for the "SercosAdrSource" input and the Sercos address for "SercosAdrDestination".

Backup with	Restore with	Remarks
	Only **MLC**: ML_AxisRestore	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
	Only **MLC**: ML_AxisRestore_Type02	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.

Tab. 11-38: Compatibility between the individual backup and restore option

Error codes

The function block uses the error table **F_RELATED_TABLE**.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Table	Additional1	Additional2	Description
STATE_MACHINE_ERROR	F_RELATED_TABL	16#0006	16#xxxx	Invalid state of the function block. Additional2 indicates an internal error number.
INPUT_RANGE_ERROR	SERCOS_TABLE	16#0002	16#0001	Invalid "BusMaster" input
INPUT_RANGE_ERROR	SERCOS_TABLE	16#0002	16#0002	Invalid "SercosAdr" input
INPUT_RANGE_ERROR	F_RELATED_TABLE	16#1E00	16#0000	The "BackupOptions" input has an invalid value.
RESOURCE_ERROR	F_RELATED_TABLE	16#1E03	16#0000	Error when reading a parameter. For further information, refer to the error structure "BackupErrors".
OTHER_ERROR	F_RELATED_TABLE	16#1E04	16#0000	Error when reading a parameter list. The list is too big to be saved in the function block.
RESOURCE_ERROR	F_RELATED_TABLE	16#1E05	16#0000	"BackupErrors" is 0 or "BackupErrorsNbr" is 0.
ACCESS_ERROR	F_RELATED_TABLE	16#1E10	16#0000	Error while creating the parameter file.
ACCESS_ERROR	F_RELATED_TABLE	16#1E12	16#0000	Error when writing the parameter file.
ACCESS_ERROR	F_RELATED_TABLE	16#1E14	16#0000	Error when closing the parameter file.

Tab. 11-39: Error codes of the IL_SIIIParameterBackup function block

Example code

A sample code shows the principle mode of operation in the following. The example is incomplete. Especially, the troubleshooting is missing. It should be used to be copied directly and then to make changes subsequently.

Declaration

```
VAR CONSTANT
  // Number of maximum errors to display
  MAX_ERRORS: UINT := 50;
END_VAR
VAR
  // Backup function block
  fbParBackup :   IL_SIIIParameterBackup;
  // Error strukture for function block
  rBackupErrors:  ARRAY [1..MAX_ERRORS] OF IL_BACKUP_ERRORS;
  // ID number of SERCOS master 0
```

```
  rBusMaster:     IL_BUSMASTER := IL_BUSMASTER_0;
  // File name, use complete path if available on your target
  strFile:        STRING := 'backup.par';
  // Defines which parameters to backup
    rOptions:       IL_BACKUP_OPTIONS := TO_SAVE_PARAM;
  // Progress of backup in %
  rBackupProgress:REAL;
END_VAR
```

Implementation

```
// Function block call
fbParBackup(Execute:= TRUE, Done=> , Active=> , Error=> , ErrorID=>,
  ErrorIdent=>, BusMaster:= rBusMaster, SercosAdr:= 1,
  BackupOptions:= rOptions, SafeAll:= FALSE, BackupFileName:= strFile,
  BackupErrors:= ADR(rBackupErrors), BackupErrorsNbr:= MAX_ERRORS,
  ExpectedCount=> , CurrentCount=> );

// Get progress in %.
// Note, that ExpectedCount will be 0 during initalisation
IF fbParBackup.ExpectedCount <> 0 THEN
  rBackupProgress := (fbParBackup.CurrentCount * 100)
                     / fbParBackup.ExpectedCount;
ELSE
  rBackupProgress := 0;
END_IF
```

11.4.2 IL_SIIIParameterRestore

Brief description

The function block IL_SIIIParameterRestore imports the S-parameters and the P-parameters of a Sercos device from a text-based file. The file can be created using the IL_SIIIParameterBackup function block (chapter 11.4.1 "IL_SIIIParameterBackup" on page 362).

The function block is provided with a similar interface as the function blocks ML_AxisRestore and ML_AxisRestore_Type02, but it is only based on the communication mechanisms of the Sercos field bus. This can backup S-parameters and P-parameters, but any number of bus devices, such as I/O devices.

Also refer to

- ML_AxisRestore
- ML_AxisRestore_Type02

Assignment: Target system/library

Target system	Library/folder
MLC	RIL_SercosIII / BackupRestore
MTX	RIL_SercosIII / BackupRestore
MLD (IL 2G)	RIL_SercosIII / BackupRestore

Tab. 11-40: Reference table of the IL_SIIIParameterRestore

Interface description

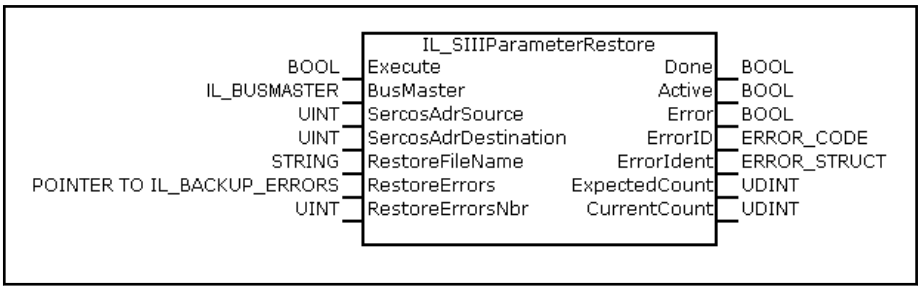


Fig. 11-14: L_SIIIParameterRestore function block

	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Processing of the function block enabled (edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SercosAdrSource	UINT	Sercos address of the slave device to import the data from. For an MLD, the value 0 has to be used as Sercos address for the local axis. **Background information**: There can be backups of multiple devices in one parameter file. The source is specified in the parameter file.
	SercosAdrDestination	UINT	Sercos address of the Slave device to import the data to. For an MLD, the value 0 has to be used as Sercos address for the local axis.
	RestoreFileName	STRING	File name of the parameter file to be imported. Specify the complete path for file systems with different paths.
	RestoreErrors	POINTER TO IL_BACKUP_ERRORS	Pointer to an array of IL_BACKUP_ERRORS structures chapter 11.4.3 "IL_BACKUP_ERRORS" on page 374
	RestoreErrorsNbr	UINT	Number of arrays in the “RestoreErrors” array
VAR_OUTPUT	Done	BOOL	Processing completed, output data valid
	Active	BOOL	Processing running, output data valid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Error diagnostics (refer to Error codes, page 372)
	ExpectedCount	UDINT	Number of parameters expected for import
	CurrentCount	UDINT	Number of already imported parameters

Tab. 11-41: Interface variables of the IL_SIIIParameterRestore 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	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	USINT	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SercosAdrSource	UINT	1	511 (MLD) 253 (other systems)	0	Rising edge at “Execute”
SercosAdrDestination	UINT	0 (MLD: local axis) 1 (other systems)	511 (MLD) 253 (other systems)	0	Rising edge at “Execute”

Name	Type	Min. value	Max. value	Default value	Effective
RestoreFile-Name	STRING	At least one letter	Up to 80 letters including path specification	-	Rising edge at “Execute”
RestoreErrors	POINTER TO IL_BACK-UP_ERRORS	> 0			Continuous
RestoreEr-rorsNbr	UINT	0	65535	0	Continuous

Tab. 11-42: Min./max. and default values of the function block IL_SIIIParameter-Restore

Functional description

The function block IL_SIIIParameterRestore imports the S-parameters and the P-parameters of a Sercos device from a text-based file. The file can be created using the IL_SIIIParameterBackup function block.

BusMaster

This is the Sercos master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0.

SercosAdrSource

Specify the Sercos address of the device, whose parameters are to be imported. The smallest possible Sercos address is MB_MIN_SERCOS_ADDRESS and the highest possible address is MB_MAX_SERCOS_ADDRESS. Depending on the target system , these definitions are provided with different values.

SercosAdrDestination

Specify the Sercos address of the device, to which the parameters are to be imported. For an MLD, the address 0 has to be specified for the local axis. Otherwise, the smallest possible Sercos address is MB_MIN_SERCOS_ADDRESS and the highest possible address is MB_MAX_SERCOS_ADDRESS. Depending on the target system , these definitions are provided with different values.

RestoreFileName

Specify the file name on the memory medium via "RestoreFileName". Always specify the complete path. Observe the case sensitivity.

RestoreErrors

This input has to be assigned with a pointer on an IL_BACKUP_ERRORS array. The structure consists of a parameter number (MB_IDN) and of an error code (DWORD).

This array lists all errors occurring during the parameter file import. It has to be distinguished between errors occurring in the function block (incorrectly input values applied) and errors resulting while writing parameters. On the latter ones mentioned are entered into this structure. The entered error numbers originate from the function blocks IL_SIIISvcWrite and IL_SIIISvcRead from the SERCOS_TABLE.

At the error output of the function block, a group error is entered in case of one entry in the structure (Additional1 = 16#1E03). This is caused, as writing a parameter is not canceled when an error results. The next parameter is written instead.

RestoreErrorsNbr

The number of structures in the "RestoreErrors" array is specified. It is recommended to use the "sizeof" operator.

ExpectedCount

This output indicates the total number of parameters to be written. The output is set to 0 in the initialization phase of the function block and is set to its final value while processing. That is, as long as the value is 0, no value has yet been determined.

CurrentCount

This output indicates the number of already read parameters. The output is set to 0 in the initialization phase of the function block and is updated while processing the function block. The value can be used to implement a progress display.

Time response

The function block IL_SIIIParameterRestore works asynchronously. Due to a rising edge at the "Execute" input after the start, it reads the inputs "BusMaster", "SercosAdrSource" and "SercosAdrDestination" and "RestoreFileName" once and the inputs "RestoreErrors" and "RestoreErrorsNbr" continuously. "Active" is set as long as the function block is operating. After the successful operation, the "Done" output is set. If an error occurs, the "Error" output is set.



Do not use the function block in the Motion task, as it sometimes requires longer execution times.

Compatibility with other parameter files

The function block can process the files required by the IL_SIIIParameterBackup function block. There is also additional information to the parameters in the parameter file that can be different depending on the backup method. The main difference is that the Sercos address is addressed for this function block, but when backing up with IndraWorks, the logical address is addressed instead.

The following table provides an overview on the different backup variants and special features to be considered.

Compatibility between the individual backup/restore function blocks

Backup with	Restore with	Remarks
IL_SercosIIIParameterBackup IndraWorksDS	IL_SercosIIIParameterRestore	✓ Parameter file can be used directly. Both function blocks use the Sercos address as addressing type.
	Only **MLC**: ML_AxisRestore	(✓) Works only if the Sercos address is equal to the logical address.
	Only **MLC**: ML_AxisRestore_Type02	Import does not work. An error is generated (Additional1 = 0x00001E16, Additional2 = 0x000A0008)

Backup with	Restore with	Remarks
IndraWorks	IL_SercosIIIParameterRestore	(✓) When importing, the logical address has to be specified for the "SercosAdrSource" input and the Sercos address for "SercosAdrDestination".
	Only **MLC**: ML_AxisRestore	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
	Only **MLC**: ML_AxisRestore_Type02	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
Only **MLC**: ML_AxisBackup	IL_SercosIIIParameterRestore	Import does not work. An error is generated (Additional1 = 0x00001E03, Additional2 = 0x0000000). Errors with different numbers are entered into the error structure.
	Only **MLC**: ML_AxisRestore	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
	Only **MLC**: ML_AxisRestore_Type02	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
Only **MLC**: ML_AxisRestore_Type02	IL_SercosIIIParameterRestore	(✓) When importing, the logical address has to be specified for the "SercosAdrSource" input and the Sercos address for "SercosAdrDestination".
	Only **MLC**: ML_AxisRestore	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.
	Only **MLC**: ML_AxisRestore_Type02	✓ Parameter file can be used directly. Both function blocks use the logical address as addressing type.

Tab. 11-43: Compatibility between the individual backup and restore option

Error codes

The function block uses the error tables **F_RELATED_TABLE** and**SERCOS_TABLE**.

It can generate the following error messages in Additional1/Additional2. Additional errors can occur due to subordinate function blocks. Its error numbers can be listed in the error table F_RELATED_TABLE.

ErrorID	Table	Additional1	Additional2	Description
STATE_MACHINE_ERROR	F_RELATED_TABLE	16#0006	16#xxxx	Invalid state of the function block. Additional2 indicates an internal error number.
INPUT_RANGE_ERROR	SERCOS_TABLE	16#0002	16#0001	Invalid "BusMaster" input

ErrorID	Table	Additional1	Additional2	Description
INPUT_RANGE_ERROR	SERCOS_TABLE	16#0002	16#0002	Invalid "SercosAdrSource" input
INPUT_RANGE_ERROR	SERCOS_TABLE	16#0002	16#0003	Invalid "SercosAdrDestination" input
RESOURCE_ERROR	F_RELATED_TABLE	16#1E03	16#0000	Error when writing a parameter. For further information, refer to the error structure "RestoreErrors".
RESOURCE_ERROR	F_RELATED_TABLE	16#1E05	16#0000	"RestoreErrors" is 0 or "RestoreErrorsNbr" is 0
ACCESS_ERROR	F_RELATED_TABLE	16#1E11	16#0000	Error when opening the parameter file
ACCESS_ERROR	F_RELATED_TABLE	16#1E13	16#0000	Error when reading the parameter file
ACCESS_ERROR	F_RELATED_TABLE	16#1E14	16#0000	Error when closing the parameter file
OTHER_ERROR	F_RELATED_TABLE	16#1E15	16#xxxxxxxx	Error when interpreting the parameter file. The line number in the parameter file is specified in Additional2.

Tab. 11-44: Error codes of the IL_SIIIParameterRestore function block

Example code

A sample code shows the principle mode of operation in the following. The example is incomplete. Especially, the troubleshooting is missing. It should be used to be copied directly and then to make changes subsequently.

Declaration

```
VAR CONSTANT
  // Number of maximum errors to display
  MAX_ERRORS: UINT := 50;
END_VAR
VAR
  // Restore function block
  fbParRestore :  IL_SIIIParameterRestore;
  // Error structure for function block
  // (will contain parameter numbers and error codes)
  rRestoreErrors: ARRAY [1..MAX_ERRORS] OF IL_BACKUP_ERRORS;
  // ID number of Sercos master 0
  rBusMaster:     IL_BUSMASTER := IL_BUSMASTER_0;
  // File name, use complete path if available on your target
  strFile:        STRING := 'backup.par';
  // Progress of backup in %
    rRestoreProgress:REAL;
END_VAR
```

Implementation

```
// Function block call
fbParRestore(Execute:= TRUE, Done=> , Active=> , Error=> , ErrorID=>,
  ErrorIdent=> , BusMaster:= rBusMaster, SercosAdrSource:= 1,
  SercosAdrDestination:= 1,
    RestoreFileName:= strFile, RestoreErrors:= ADR(rRestoreErrors),
    RestoreErrorsNbr:= MAX_ERRORS, ExpectedCount=> , CurrentCount=> );

// Get progress in %.
// Note, that ExpectedCount will be 0 during initalisation
IF fbParRestore.ExpectedCount <> 0 THEN
  rRestoreProgress := (fbParRestore.CurrentCount * 100)
                      / fbParRestore.ExpectedCount;
ELSE
  rRestoreProgress := 0;
END_IF
```

11.4.3 IL_BACKUP_ERRORS

Type definitions

The following definition describes the data type IL_BACKUP_ERRORS used in the following function blocks.

- IL_SIIIParameterBackup, page 362
- IL_SIIIParameterRestore, page 368

The type definitions are included in the library.

The data type "IL_BACKUP_ERRORS" is defined as follows:

Structural element	Data type	Description
ParameterNumber	MB_IDN	Parameter number in IDN format
ErrorCode	DWORD	Error code

Tab. 11-45: Defining the IL_BACKUP_ERRORS data type

11.4.4 IL_BACKUP_OPTIONS

Type definitions

The following definition describes the enumeration type IL_BACKUP_OPTIONS used in the following function block.

- IL_SIIIParameterBackup, page 362

The type definitions are included in the library.

The data type "IL_BACKUP_OPTIONS" is defined as follows:

Enumeration element	Value	Description
ALL_PARAM	1	All parameters
MODIFIED_PARAM	2	Parameters deviating from the default value
TO_SAVE_PARAM	3	Parameters required for restoration

Tab. 11-46: Defining the enumeration elements of the IL_BACKUP_OPTIONS data type

11.5 Diagnostics

11.5.1 IL_SIIIGetConfigSlaveAdrList

Brief description

Use the IL_SIIIGetConfigSlaveAdrList function block for a list of the Sercos addresses of all configured PLC Sercos devices.

Target system	Library/folder
MLC	RIL_SercosIII/Diagnostics
MTX	RIL_SercosIII/Diagnostics
MLD (IL 2G)	RIL_SercosIII/Diagnostics

Tab. 11-47: Reference table

Interface description

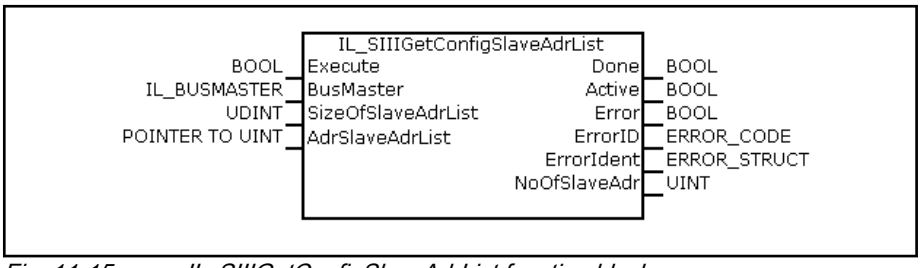


Fig. 11-15: IL_SIIIGetConfigSlaveAdrList function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the "BusMaster" input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SizeOfSlaveAdrList	UDINT	Size in bytes of buffer at the "ValueAdr" address in bytes provided for data reception (SIZEOF(SlaveAdrList))
	AdrSlaveAdrList	POINTER TO UINT	Buffer address provided for the list of slave addresses (ADR(SlaveAdrList)).
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	NoOfSlaveAdr	UINT	Number of bytes copied to the "ValueAdr" buffer.

Tab. 11-48: IL_SIIIGetConfigSlaveAdrList interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SizeOfSlaveAdrList	UDINT	0	16#FFFFFFFF	0	Rising edge at “Execute”
AdrSlaveAdrList	POINTER TO UINT	N. def.	N. def.	0	Rising edge at “Execute”

Tab. 11-49: Min./max. and default values of the IL_SIIIGetConfigSlaveAdrList inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found in error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0002	16#0001	The input value of the parameter "BusMaster" is unequal to IL_BUSMASTER_0 and outside of the input value range.
INPUT_RANGE_ERROR	16#0002	16#0006	The input value of the parameter "AdrSlaveAdrList" is 0

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#0006	16#0001	Error in state machine
INPUT_INVALID_ERROR	16#2907	Required value for "SizeOfSlaveAdrList"	The input value of the parameter "SizeOfSlaveAdrList" is smaller than required

Tab. 11-50: Error codes of function block IL_SIIIGetConfigSlaveAdrList in F_RELATED_TABLE := 16#0170

11.5.2 IL_SIIIGetDiagSlaveAdrList

Brief description Use the IL_SIIIGetDiagSlaveAdrList function block for a list of the Sercos addresses of all PLC Sercos devices for diagnostic information.

Target system	Library/folder
MLC	RIL_SercosIII/Diagnostics
MTX	RIL_SercosIII/Diagnostics
MLD (IL 2G)	RIL_SercosIII/Diagnostics

Tab. 11-51: Reference table

Interface description

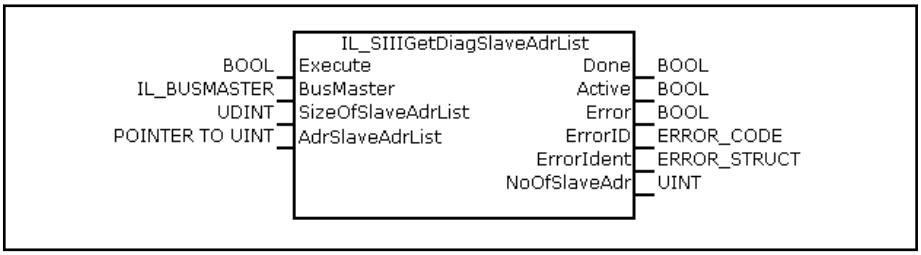


Fig. 11-16: IL_SIIIGetDiagSlaveAdrList function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the "BusMaster" input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SizeOfSlaveAdrList	UDINT	Size in bytes of buffer at the "ValueAdr" address in bytes provided for data reception (SIZEOF(SlaveAdrList))
	AdrSlaveAdrList	POINTER TO UINT	Buffer address provided for the list of slave addresses (ADR(SlaveAdrList)).
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error

I/O type	Name	Data type	Description
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	NoOfSlaveAdr	UINT	Number of bytes copied to the ValueAdr buffer.

Tab. 11-52: IL_SIIIGetDiagSlaveAdrList interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SizeOfSlaveAdrList	UDINT	0	16#FFFFFFFF	0	Rising edge at “Execute”
AdrSlaveAdrList	POINTER TO UINT	N. def.	N. def.	0	Rising edge at “Execute”

Tab. 11-53: Min./max. and default values of the IL_SIIIGetDiagSlaveAdrList inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found in error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0002	16#0001	The input value of the parameter "BusMaster" is unequal to IL_BUSMASTER_0 and outside of the input value range.
INPUT_RANGE_ERROR	16#0002	16#0006	The input value of the parameter "AdrSlaveAdrList" is 0
STATE_MACHINE_ERROR	16#0006	16#0001	Error in state machine
INPUT_INVALID_ERROR	16#2907	Required value for "SizeOfSlaveAdrList"	The input value of the parameter "SizeOfSlaveAdrList" is smaller than required

Tab. 11-54: Error codes of function block IL_SIIIGetDiagSlaveAdrList in F_RELATED_TABLE := 16#0170

11.5.3 IL_SIIIGetLiveSlaveAdrList

Brief description

Use the IL_SIIIGetLiveSlaveAdrList function block for a list of the Sercos addresses of all Sercos devices found at the bus.

Target system	Library/folder
MLC	RIL_SercosIII/Diagnostics
MTX	RIL_SercosIII/Diagnostics
MLD (IL 2G)	RIL_SercosIII/Diagnostics

Tab. 11-55: Reference table

Interface description

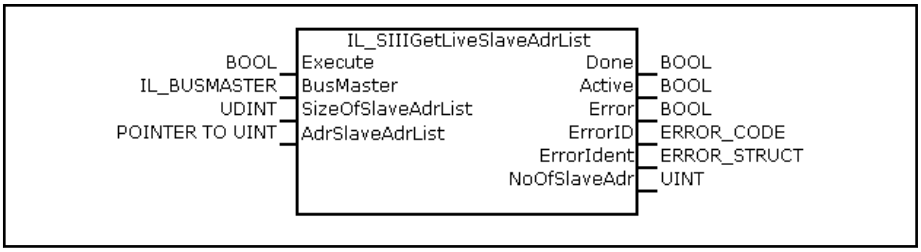


Fig. 11-17: IL_SIIIGetLiveSlaveAdrList function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the "BusMaster" input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SizeOfSlaveAdrList	UDINT	Size in bytes of buffer at the "ValueAdr" address in bytes provided for data reception (SIZEOF(SlaveAdrList))
	AdrSlaveAdrList	POINTER TO UINT	Buffer address provided for the list of slave addresses (ADR(SlaveAdrList)).
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid.
	Active	BOOL	Processing active, output data invalid.
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	NoOfSlaveAdr	UINT	Number of bytes copied to the "ValueAdr" buffer.

Tab. 11-56: IL_SIIIGetLiveSlaveAdrList interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SizeOfSlaveAdrList	UDINT	0	16#FFFFFFFF	0	Rising edge at “Execute”
AdrSlaveAdrList	POINTER TO UINT	N. def.	N. def.	0	Rising edge at “Execute”

Tab. 11-57: Min./max. and default values of the IL_SIIIGetLiveSlaveAdrList inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found in error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0002	16#0001	The input value of the parameter "BusMaster" is unequal to IL_BUSMASTER_0 and outside of the input value range.
INPUT_RANGE_ERROR	16#0002	16#0006	The input value of the parameter "AdrSlaveAdrList" is 0
STATE_MACHINE_ERROR	16#0006	16#0001	Error in state machine
INPUT_INVALID_ERROR	16#2907	Required value for "SizeOfSlaveAdrList"	The input value of the parameter "SizeOfSlaveAdrList" is smaller than required

Tab. 11-58: Error codes of function block IL_SIIIGetLiveSlaveAdrList in F_RELATED_TABLE := 16#0170

11.5.4 IL_SIIIMasterState

Brief description The IL_SIIIMasterState function block provides the status of the Sercos master.

Target system	Library/folder
MLC	RIL_SercosIII/Diagnostics
MTX	RIL_SercosIII/Diagnostics
MLD (IL 2G)	RIL_SercosIII/Diagnostics

Tab. 11-59: Reference table

Interface description

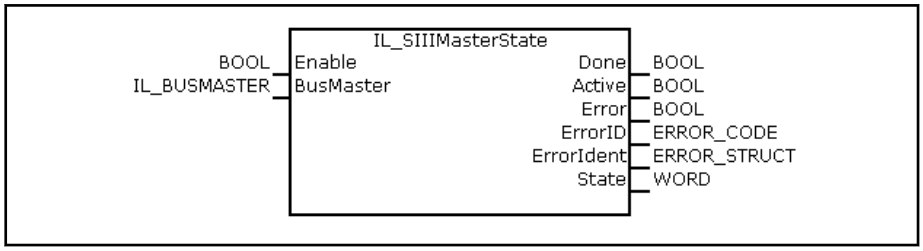


Fig. 11-18: IL_SIIIMasterState function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (permanent, level-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the "BusMaster" input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	State	WORD	Status of the Sercos master

Tab. 11-60: IL_SIIIMasterState interface

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

"State" output

Bit no.	Name	Description
0	IL_FBM_BUS_OFF	Field bus not ready. The Sercos bus is in a Sercos phase unequal to Sercos phase 4. There is no cyclic communication.
1	IL_FBM_MASTER_FAILURE	Bus master failure
2	IL_FBM_SLAVE_NOT_REACHA-BLE	One or more field bus slaves cannot be reached (slave not reachable)
3	IL_FBM_SLAVE_DIAGNOSIS	One or more field bus slaves signal diagnoses (slave diag)
4	IL_FBM_SLAVE_ERROR	One or more field bus slaves report errors (slave error)

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”

Tab. 11-61: Min./max. and default values of the IL_SIIIMasterState inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found in error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0002	16#0001	The input value of the parameter "BusMaster" is unequal to IL_BUSMASTER_0 and outside of the input value range.
STATE_MACHINE_ERROR	16#0006	16#0001	Error in state machine

Tab. 11-62: Error codes of function block IL_SIIIMasterState in F_RELATED_TA-BLE := 16#0170

11.5.5 IL_SIIIMasterStateDetails

Brief description

The IL_SIIIMasterStateDetails function block provides the detailed status of the Sercos master with error counters and topology information.

Target system	Library/folder
MLC	RIL_SercosIII/Diagnostics
MLD (IL 2G)	RIL_SercosIII/Diagnostics

Tab. 11-63: Reference table

☞ IL_SIIIMasterStateDetails is not available for the MTX system.

Interface description

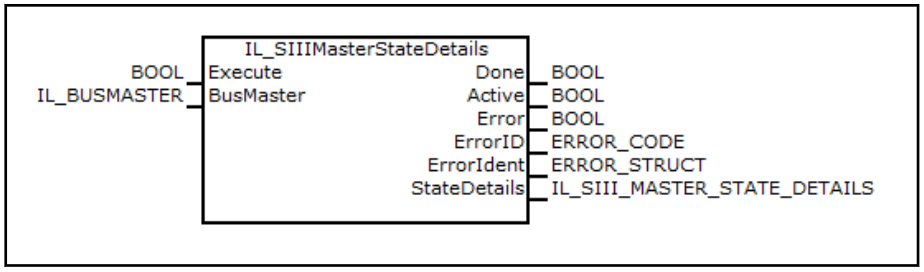


Fig. 11-19: IL_SIIIMasterStateDetails function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (permanent, level-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the "BusMaster" input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	StateDetails	IL_SIII_MASTER_STATE_DETAILS	Detailed status of the master Also refer to IL_SIII_MASTER_STATE_DETAILS, page 385

Tab. 11-64: IL_SIIIMasterStateDetails interface

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”

Tab. 11-65: Min./max. and default values of the IL_SIIIMasterStateDetails inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

The errors of the internally used function blocks IL_SIIIMasterState and IL_SIIIGetLiveSlaveAdrList as well as the following function block errors are listed in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#0006	16#0001	Error in state machine

Tab. 11-66: Error codes of function block IL_SIIIMasterStateDetails in F_RELATED_TABLE := 16#0170

11.5.6 IL_SIIIRemoteSlaveState

Brief description

The IL_SIIIRemoteSlaveState function block provides the status of a slave operated at the Sercos master.

Target system	Library/folder
MLC	RIL_SercosIII/Diagnostics
MTX	RIL_SercosIII/Diagnostics
MLD (IL 2G)	RIL_SercosIII/Diagnostics

Tab. 11-67: Reference table

Interface description

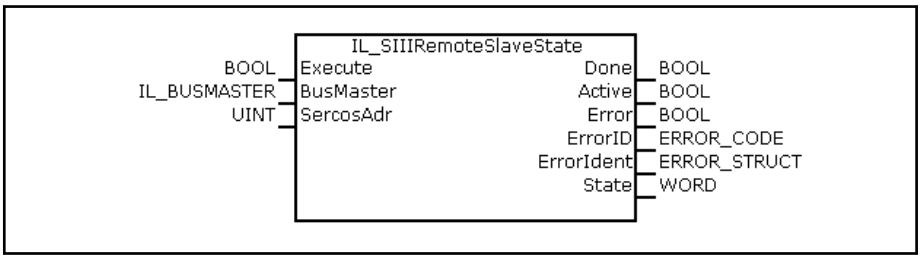


Fig. 11-20: IL_SIIIRemoteSlaveState function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the "BusMaster" input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	State	WORD	Status of a slave operated at the Sercos master

Tab. 11-68: IL_SIIIRemoteSlaveState interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMAS-TER_0	Rising edge at “Execute”
SercosAdr	UINT	1	511	0	Rising edge at “Execute”

Tab. 11-69: Min./max. and default values of the IL_SIIIRemoteSlaveState inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found in error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0002	16#0001	The input value of the parameter "BusMaster" is unequal to IL_BUSMASTER_0 and outside of the input value range.
INPUT_RANGE_ERROR	16#0002	16#0002	The input value of the parameter "SercosAdr" is 0 or >511
STATE_MACHINE_ERROR	16#0006	16#0001	Error in state machine
INPUT_INVALID_ERROR	16#2906	16#0000	Input value of the parameter "SercosAdr" addresses a slave that is not present in the configuration.

Tab. 11-70: Error codes of function block IL_SIIIRemoteSlaveState in F_RELATED_TABLE := 16#0170

11.5.7 IL_SIIIRemoteSlaveStateDetails

Brief description

The IL_SIIIRemoteSlaveStateDetails function block provides the (detailed) status of a slave operated at the Sercos master.

Target system	Library/folder
MLC	RIL_SercosIII/Diagnostics
MTX	RIL_SercosIII/Diagnostics
MLD (IL 2G)	RIL_SercosIII/Diagnostics

Tab. 11-71: Reference table

Interface description

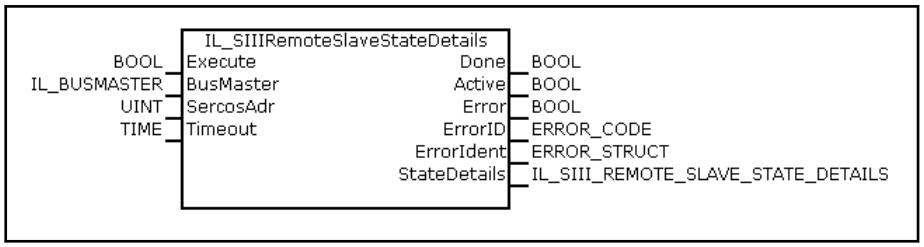


Fig. 11-21: IL_SIIIRemoteSlaveStateDetails function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the "BusMaster" input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	Timeout	TIME	Timeout monitoring of the function block in ms. T#0ms = Timeout monitoring not active.
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics
	StateDetails	IL_SIII_REMOTE_SLAVE_STATE_DETAILS	Detailed status of a slave operated at the Sercos master IL_SIII_REMOTE_SLAVE_STATE_DETAILS, page 385

Tab. 11-72: IL_SIIIRemoteSlaveStateDetails interface

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	BYTE	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at “Execute”
SercosAdr	UINT	1	511	0	Rising edge at “Execute”

Tab. 11-73: Min./max. and default values of the IL_SIIIRemoteSlaveStateDetails inputs

Troubleshooting

This function block uses the error table SERCOS_TABLE (ERROR_TABLE = 16#0010) to represent errors of the Sercos service channel.

In addition, the following errors of the function block can be found in error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#0002	16#0001	The input value of the parameter "BusMaster" is unequal to "IL_BUSMASTER_0" and outside of the input value range.
INPUT_RANGE_ERROR	16#0002	16#0002	The input value of the parameter "SercosAdr" is 0 or >511

ErrorID	Additional1	Additional2	
STATE_MACHINE_ERROR	16#0006	16#0001	Error in state machine
INPUT_INVALID_ERROR	16#2906	16#0000	Input value of the parameter "SercosAdr" addresses a slave that is not present in the configuration.

Tab. 11-74: Error codes of function block IL_SIIIRemoteSlaveStateDetails in F_RELATED_TABLE := 16#0170

11.5.8 IL_SIII_MASTER_STATE_DETAILS

Brief description The structure contains the detailed status of a Sercos master.

Structural element	Data type	Description
CommunicationCOS	DWORD	Refer to the output State output, page 380 of the function block "IL_SIII-MasterState".
CommunicationState	IL_FBUS_COMMU-NICATION_STATE	Status of the communication. CP0/CP1=Not configured | CP2=STOP | CP3=IDLE |CP4=OPERATE. IL_FBUS_COMMUNICATION_STATE, page 185
Version	UINT	Version of the data structure of the diagnostic information
ErrorCounterPri	UDINT	Error counter, primary (port 1, port names of different controls) of the control
ErrorCounterSec	UDINT	Error counter, secondary (port 2, port names of different controls) of the control
NumOfActiveSlaves	UDINT	Number of active slaves, i.e., slaves to which the master has established a communication connection
SercosAdrEndLine1	UINT	Sercos address of the slave at the end of the line at port 1 (refer to the following table, port names of the different controls). 0 = Ring topology or no device is connected at this line
SercosAdrEndLine2	UINT	Sercos address of the slave at the end of the line at port 2 (refer to the following table, port names of the different controls). 0 = Ring topology or no device is connected at this line
RingbreakTopoAdr	UINT	Topological address of the ring break 16#FFFF = Ring topology

Tab. 11-75: Elements of structure IL_SIII_MASTER_STATE_DETAILS

Port	XM	VPx	MLD-M (CSH02, HCS01)
1	XF1	XF01	X25
2	XF2	XF02	X24

Tab. 11-76: Port names of different controls

11.5.9 IL_SIII_REMOTE_SLAVE_STATE_DETAILS

Brief description The structure contains the detailed status of a slave operated at the Sercos master

The following table shows its substructures:

Substructures and their dependencies

IL_SIII_REMOTE_SLAVE_STATE_EX, page 388		
	IL_SIII_REMOTE_SLAVE_STATE_EX_BITS, page 388	
IL_SIII_DEVICE_STATUS, page 389		
	IL_SIII_DEVICE_STATUS_UNION, page 389	
		IL_SIII_DEVICE_STATUS_UNION_BITS, page 390
	IL_SIII_TOPOLOGY_PORT_STATUS, page 390	
IL_SIII_DIAGNOSTIC_NUMBER, page 391		
	IL_SIII_DIAGNOSTIC_NUMBER_R, page 392	
		IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP, page 392
		IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE, page 392
		IL_SIII_DIAGNOSTIC_NUMBER_CLASS, page 393
IL_SIII_DEVICE_TYPE, page 393		
IL_SIII_DRIVE_DIAG, page 393		
	IL_SIII_DRV_ERROR, page 394	
		IL_SIII_DRV_ERROR_BITS, page 394
	IL_SIII_DRV_WARNING, page 395	
		IL_SIII_DRV_WARNING_BITS, page 395
	IL_SIII_DRV_MESSAGE, page 396	
		IL_SIII_DRV_MESSAGE_BITS, page 396
IL_SIII_IO_DIAG, page 397		
	IL_SIII_IO_DIAG_STATUS_CODE_TYPE, page 397	
	IL_SIII_DIAGNOSTIC_NUMBER_CLASS, page 393	
	IL_SIII_IO_DIAG_STATUS_CODE, page 398	

Structural element	Data type	Description
MasterDiagNo	UDINT	Master diagnostic number, which the master reports for the slave. WRONG_VENDORCODE 1 The master detects in Sercos phase 2 that the vendor ID of the device description of the slave does not match the vendor ID (VendorCode S-1300.0.3) read by the slave. WRONG_DEVICEID 2 The master detects in Sercos phase 2 that the device ID of the device description of the slave does not match the vendor ID (VendorCode S-1300.00.5) read by the slave. WRONG_MODULENBR 3 The master detects in Sercos phase 2 that the number of I/O modules specified in the IndraWorks project does not match the modules determined at the slave (list of module type codes S-0-1500.x.03). WRONG_MODULEID 4 The master detects in Sercos phase 2 that a module detection specified in the IndraWorks project does not match the module detection determined at the slave (list of module type codes S-0-1500.x.03). SLAVEERROR 5 Slave device reports an error via the bus. Refer to the structural element DiagNumber. SLAVEWARN 6 Slave device reports an error via the bus. Refer to the structural element DiagNumber. ERR_SWITCH_P3 7 Error when switching from Sercos phase 2 to Sercos phase 3. ERR_SWITCH_P4 8 Error when switching from Sercos phase 2 to Sercos phase 3. ERR_REPLACEFG 9 Error in mechanism "Replace Function Groups". ERR_WRITEPARAM_P2 16#A Error when writing a parameter in phase 2. ERR_WRITEPARAM_P3 16#B Error when writing a parameter in phase 3. ERR_WRITEPARAM_P4 16#C Error when writing a parameter in phase 4. WRONG_ATLENGTH 16#D Incorrect AT length. WRONG_MDTLENGTH 16#E Incorrect MDT length. CON_TIMEOUT 16#F Master detects connection timeout. CON_ERROR 16#10 Slave detects connection error. NEWCONFIG 16#11 New IndraWorks configuration available, but not yet active. Bus restart required. NOT_ON_BUS 16#12 No slave at the bus.

Structural element	Data type	Description
MasterDiagNoEx	UDINT	Additional error number for MasterDiagNo
MasterDiagText	STRING	Note text on master diagnostic number (only English)
MasterDiagSolutionText	STRING	Text providing information on troubleshooting the error image (only English)
StateEx	IL_SIII_REMOTE_SLAVE_STATE_EX	All errors present for the slave are bit-coded IL_SIII_REMOTE_SLAVE_STATE_EX, page 388
DeviceStatus	IL_SIII_DEVICE_STATUS	Corresponds to parameter device status S-0-1045 IL_SIII_DEVICE_STATUS, page 389
DiagNumber	IL_SIII_DIAGNOSTIC_NUMBER	Corresponds to Sercos parameter S-0-0390 of the Sercos slave IL_SIII_DIAGNOSTIC_NUMBER, page 391
DiagText	STRING	Corresponds to Sercos parameter diagnostic text S-0-0095 of the Sercos slave
DeviceType	IL_SIII_DEVICE_TYPE	Indicates device type of slave IL_SIII_DEVICE_TYPE, page 393
DriveDiag	IL_SIII_DRIVE_DIAG	Indicates drive diagnostic information IL_SIII_DRIVE_DIAG, page 393
IoDiag	IL_SIII_IO_DIAG	Indicates I/O diagnostic information IL_SIII_IO_DIAG, page 397

11.5.10 IL_SIII_REMOTE_SLAVE_STATE_EX

Brief description The IL_SIII_REMOTE_SLAVE_STATE_EX structure contains all errors present for the slave as bit-coded errors

	Data type	Description
StateBits	IL_SIII_REMOTE_SLAVE_STATE_EX_BITS	Bit-coded DWORD displaying different error states and status. IL_SIII_REMOTE_SLAVE_STATE_EX_BITS, page 388
StateDWord	DWORD	

Tab. 11-77: Elements of the IL_SIII_REMOTE_SLAVE_STATE_EX structure

11.5.11 IL_SIII_REMOTE_SLAVE_STATE_EX_BITS

Brief description The IL_SIII_REMOTE_SLAVE_STATE_EX_BITS structure contains all errors possible for the slave as bit-coded DWORD displaying different error states and status

Structural element	Type	Description
NewConfig	BIT	Bit-coded DWORD displaying different error states and status
NotOnBus	BIT	No slave at the bus.
ErrVendorCode	BIT	The master detects in Sercos phase 2 that the vendor ID of the device description of the slave does not match the vendor ID (VendorCode S-1300.0.3) read by the slave.
ErrDeviceId	BIT	The master detects in Sercos phase 2 that the device ID of the device description of the slave does not match the vendor ID (VendorCode S-1300.00.5) read by the slave.

Structural element	Type	Description
ErrModuletypeNbr	BIT	The master detects in Sercos phase 2 that the number of I/O modules specified in the IndraWorks project does not match the modules determined at the slave (list of module type codes S-0-1500.x.03).
ErrModuletype	BIT	The master detects in Sercos phase 2 that a module detection specified in the IndraWorks project does not match the module detection determined at the slave (list of module type codes S-0-1500.x.03).
ErrReplaceFg	BIT	Error in mechanism "Replace Function Groups".
ErrParamP2	BIT	Error when writing a parameter in phase 2.
ErrParamP3	BIT	Error when writing a parameter in phase 3.
ErrMdtLength	BIT	Incorrect MDT length
ErrAtLength	BIT	Incorrect AT length
ErrCl1Err	BIT	Slave device reports an error via the bus. Refer to the structural element DiagNumber.
ErrCl2Warn	BIT	Slave device reports an error via the bus. Refer to the structural element DiagNumber.
ConMdtErr	BIT	Slave detects connection error.
ConAtTimeout	BIT	Master detects connection timeout.
ConOff	BIT	At least one Sercos connection is not in the "consuming" state

Tab. 11-78: Elements of the IL_SIII_REMOTE_SLAVE_STATE_EX_BITS structure

11.5.12 IL_SIII_DEVICE_STATUS

Brief description The IL_SIII_DEVICE_STATUS structure corresponds to the parameter device status S-0-1045 supplemented by the topology status information of both ports.

Structural element	Data type	Description
Status	IL_SIII_DEVICE_STATUS_UNION	Bit structure of the Sercos parameter S-0-1045, Device status (S-Dev) IL_SIII_DEVICE_STATUS_UNION, page 389
TopologyPortStatus	IL_SIII_TOPOLOGY_PORT_STATUS	Support the status display of both ports IL_SIII_TOPOLOGY_PORT_STATUS, page 390

Tab. 11-79: Elements of the IL_SIII_DEVICE_STATUS structure

11.5.13 IL_SIII_DEVICE_STATUS_UNION

Brief description The IL_SIII_DEVICE_STATUS_UNION union superimposes the status bits and the status word of the parameter "Device Status S-0-1045" for a more comfortable data access.

Element	Data type	Description
StatusBits	IL_SIII_DEVICE_STATUS_UNION_BITS	Bit structure of the Sercos parameter S-0-1045, Device status (S-Dev) IL_SIII_DEVICE_STATUS_UNION_BITS, page 390
StatusWord	WORD	

Tab. 11-80: Elements of the UNION IL_SIII_DEVICE_STATUS_UNION

11.5.14 IL_SIII_DEVICE_STATUS_UNION_BITS

Brief description The IL_SIII_DEVICE_STATUS_UNION_BITS structure contains the individual bits of the parameter "Device Status S-0-1045".

Structural element	Type	Description
CommunicationWarning	BIT	The number of permitted MST failures exceeded half the failures specified in the parameter S-0-1003
TopologyHs	BIT	Toggling the bit signals a change in topology status
TopologPortStatus_3	BIT	
TopologPortStatus_2	BIT	
TopologPortStatus_1	BIT	
TopologPortStatus_0	BIT	
ErrorConnection	BIT	The slave detected an error in the connection
SlaveValid	BIT	The "DeviceStatus" contents are valid
ErrorOfDevice	BIT	Slave reports an error
WarningOfDevice	BIT	Slave reports a warning
ProcedureCommandAck	BIT	Acknowledging a procedure command
ParameterizationLevel	BIT	Parameterization level active
LoopbackWithoutFwd	BIT	Sercos III telegram is not transferred to the inactive port
Reserved_2	BIT	Reserved
Reserved_1	BIT	Reserved
Reserved_0	BIT	Reserved

Tab. 11-81: Elements of the IL_SIII_DEVICE_STATUS_UNION_BITS structure

11.5.15 IL_SIII_TOPOLOGY_PORT_STATUS

Brief description The enumeration type IL_SIII_TOPOLOGY_PORT_STATUS (UINT) support the status display of both ports.

Element	Value	Description
FAST_FORWARD_ON_BOTH_PORTS	16#00	Sercos device connected.
LOOPBACK_WITH_FORWARD_OF_P_TELEGRAMS_WITHOUT_LINK	16#04	No link at inactive port. No device connected.
LOOPBACK_WITHOUT_FORWARD_OF_P_TELEGRAMS_WITHOUT_LINK	16#1C	No link at inactive port. No device connected.

Element	Value	Description
LOOPBACK_WITH_FORWARD_OF_P_TELEGRAMS_WITH_LINK	16#05	Link at inactive port. No device connected.
LOOPBACK_WITHOUT_FORWARD_OF_P_TELEGRAMS_WITH_LINK	16#1D	Link at inactive port. No device connected.
LOOPBACK_WITH_FORWARD_OF_P_TELEGRAMS_WITH_P_LINK	16#40	Sercos device connected.
LOOPBACK_WITH_FORWARD_OF_P_TELEGRAMS_WITH_S_LINK	16#41	Sercos device connected.
LOOPBACK_WITHOUT_FORWARD_OF_P_TELEGRAMS_WITH_S_LINK	16#42	Sercos device connected.
LOOPBACK_WITH_FORWARD_OF_S_TELEGRAMS_WITHOUT_LINK	16#43	No link at inactive port. No device connected
LOOPBACK_WITHOUT_FORWARD_OF_S_TELEGRAMS_WITHOUT_LINK	16#44	No link at inactive port. No device connected
LOOPBACK_WITH_FORWARD_OF_S_TELEGRAMS_WITH_LINK	16#45	Link at inactive port. No device connected
LOOPBACK_WITHOUT_FORWARD_OF_S_TELEGRAMS_WITH_LINK	16#46	Link at inactive port. No device connected
LOOPBACK_WITH_FORWARD_OF_S_TELEGRAMS_WITH_P_LINK	16#47	Sercos device connected
LOOPBACK_WITHOUT_FORWARD_OF_S_TELEGRAMS_WITH_P_LINK	16#48	Sercos device connected
LOOPBACK_WITH_FORWARD_OF_S_TELEGRAMS_WITH_S_LINK	16#49	Sercos device connected

Tab. 11-82: Elements of enumeration type IL_SIII_TOPOLOGY_PORT_STATUS

11.5.16 IL_SIII_DIAGNOSTIC_NUMBER

Brief description The superimposition type IL_SIII_DIAGNOSTIC_NUMBER supports the access to the individual sections of the complete Sercos diagnostic number (S-0-0390). The user can also access unstructured information as DWORD.

Structural element	Data type	Description
Number	IL_SIII_DIAGNOSTIC_NUMBER_R	Corresponds to parameter S-0-0390. IL_SIII_DIAGNOSTIC_NUMBER_R, page 392
NumberDWord	DWORD	

Tab. 11-83: Elements of the superimposition type IL_SIII_DIAGNOSTIC_NUMBER

The current diagnostic number is displayed in the parameter S-0-0390. Thus, the control can generate individual diagnostics using this number (e.g. in languages in which the diagnostic texts are not saved in the drive).

From MPx-18VRS, Sercos-compliant diagnostic codes are used (bit 24 to 31). To achieve a compatible behavior between previous versions and the MPx-18VRS and higher, bit 0 has to be set to "1" in the parameter P-0-0006.

11.5.17 IL_SIII_DIAGNOSTIC_NUMBER_R

Brief description Individual elements of the complete diagnostic number can be accessed with IL_SIII_DIAGNOSTIC_NUMBER_R.

Structural element	Data type	Description
SourceGroup	IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP	Describes the use of bits of the complete diagnostic number IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP, page 392
SourceType	IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE	Diagnostic source IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE, page 392
Class	IL_SIII_DIAGNOSTIC_NUMBER_CLASS	Priority of the present diagnostics IL_SIII_DIAGNOSTIC_NUMBER_R, page 392
StatusCode	UINT	Error number

Tab. 11-84: Elements of the IL_SIII_DIAGNOSTIC_NUMBER_R structure

11.5.18 IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP

Brief description The enumeration type IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP describes the use of bits of the complete diagnostic number.

Element	Value	Description
MANUFACTURER_SPECIFIC_15_0	0	Manufacturer-specific status code Bits 29-16 (SourceType and Class) defined by Sercos Bits 15-0, status code is manufacturer-specific
MANUFACTURER_SPECIFIC_29_0	1	Completely manufacturer-specific Bits 29-0 specified by the device manufacturer
RESERVED	2	Reserved
STANDARD	3	Default Bits 29-0 defined by Sercos

Tab. 11-85: Elements of the enumeration type IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_GROUP

11.5.19 IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE

Brief description The enumeration type IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE provides information on the origin of the diagnostics.

Element	Value	Description
ST_FSP_DRIVE	0	StatusCode defined by drive profile
ST_FSP_IO	1	StatusCode defined by I/O profile
ST_GDP	2	StatusCode defined by GDP profile
ST_SCP	3	StatusCode defined by SCP profile

Element	Value	Description
ST_CSoS	4	StatusCode defined by CSoS specification
ST_FSP_ENCODER	5	StatusCode defined by encoder profile
ST_SAFETY_APPLICATION	6	StatusCode defined by Safety application

Tab. 11-86: Elements of the enumeration type IL_SIII_DIAGNOSTIC_NUMBER_SOURCE_TYPE

11.5.20 IL_SIII_DIAGNOSTIC_NUMBER_CLASS

Brief description The enumeration type IL_SIII_DIAGNOSTIC_NUMBER_CLASS indicates the priorities of the diagnostics.

Element	Value	Description
RESERVED_0	16#00	Reserved: 16#00...16#09
OPERATIONAL_STATE	16#0A	Message, priority 4. Provides information on the general operating state, e.g. Drive HALT...
RESERVED_1	16#0B	Reserved
PROCEDUR_CMD_STATE	16#0C	Command error, priority 3. Indicates that an error occurred while processing a command.
RESERVED_2	16#0D	Reserved
WARNING_C2D	16#0E	Warning, priority 2
ERROR_C1D	16#0F	Error with highest priority (1). Error has to be acknowledged Also refer to S-0-0099, C0500 reset state class 1

Tab. 11-87: Elements of the enumeration type IL_SIII_DIAGNOSTIC_NUMBER_CLASS

11.5.21 IL_SIII_DEVICE_TYPE

Brief description The enumeration type IL_SIII_DEVICE_TYPE (UDINT) lists the different device types.

Element	Value	Description
FSP_IO	1	Device type acc. to I/O profile
FSP_DRIVE	2	Device type acc. to drive profile
FSP_ENCODER	3	Device type acc. to encoder profile
FSP_MANUFATURER_SPECIFIC	16#00008000	Manufacturer-specific device type
FSP_UNKNOWN	16#FFFFFFFF	Unknown device type

Tab. 11-88: Elements of enumeration type IL_SIII_DEVICE_TYPE

11.5.22 IL_SIII_DRIVE_DIAG

Brief description The IL_SIII_DRIVE_DIAG structure summarizes the possible diagnostic bits of errors, warnings and messages.

Structural element	Data type	Description
DrvError	IL_SIII_DRV_ERROR	UNION of the error bits, see S-0-0011 IL_SIII_DRV_ERROR, page 394
DrvWarning	IL_SIII_DRV_WARNING	UNION of the warning bits, see S-0-0012 IL_SIII_DRV_WARNING, page 395
DrvMessage	IL_SIII_DRV_MESSAGE	UNION of the message bits, see S-0-0013 IL_SIII_DRV_MESSAGE, page 396

Tab. 11-89: Elements of the IL_SIII_DRIVE_DIAG structure

11.5.23 IL_SIII_DRV_ERROR

Brief description The superimposition type (UNION) IL_SIII_DRV_ERROR supports access to and display of error bits acc. to S-0-0011.

Structural element	Data type	Description
ErrorBits	IL_SIII_DRV_ERROR_BITS	Corresponds to parameter S-0-0011 IL_SIII_DRV_ERROR_BITS, page 394
ErrorWord	WORD	

Tab. 11-90: Elements of the superimposition type IL_SIII_DRV_ERROR

11.5.24 IL_SIII_DRV_ERROR_BITS

Brief description The IL_SIII_DRV_ERROR_BITS structure contains the individual bits of the parameter S-0-0011.

Structural element	Type	Description
ManufacturerSpecificError	BIT	Manufacturer-specific error code, see S-0-0129
Reserved	BIT	
TravelRangeIsExceeded	BIT	Position limit values exceeded, see S-0-0049, S-0-0050 or S-0-0478, Position/ travel range limit status if activated
NotUsed	BIT	
ExcessivePositionDeviation	BIT	See S-0-0159, Monitoring window or S-0-0391, Monitoring window feedback 2 or difference between position command values too high
PowerSupplyPhaseError	BIT	Supply voltage error
UnderVoltageError	BIT	Undervoltage error, supply voltage or DC bus voltage. Limit value is manufacturer-specific
OverVoltageError	BIT	Overvoltage error, supply voltage or DC bus voltage. Limit value is manufacturer-specific
OverCurrentError	BIT	Overcurrent error
ErrorInTheCommutationSystem	BIT	Error in "commutation system"
FeedbackErrorEncoder	BIT	Motor encoder: S-0-0390, Diagnostic number, contains 0xC00F8022 External encoder: S-0-0390, Diagnostic number, contains 0xC00F2042
ControlVoltageError	BIT	Failure of control voltage, S-0-0390, Diagnostic number, contains 0xC00F8070
CoolingErrorShutDown	BIT	See S-0-0205

Structural element	Type	Description
MotorOverTemperatureShut-Down	BIT	See S-0-0204, Motor switch-off temperature exceeded
AmplifierOverTemperature-ShutDown	BIT	See S-0-0203
OverloadShutDown	BIT	See S-0-0114

Tab. 11-91: Elements of the IL_SIII_DRV_ERROR_BITS structure

11.5.25 IL_SIII_DRV_WARNING

Brief description The superimposition type (UNION) IL_SIII_DRV_WARNING supports access to and display of warning bits acc. to S-0-0012.

Structural element	Data type	Description
WarningBits	IL_SIII_DRV_WARNING_BITS	Corresponds to parameter S-0-0012 IL_SIII_DRV_WARNING_BITS, page 395
WarningWord	WORD	

Tab. 11-92: Elements of the superimposition type IL_SIII_DRV_WARNING

11.5.26 IL_SIII_DRV_WARNING_BITS

Brief description The IL_SIII_DRV_WARNING_BITS structure contains the individual bits of the parameter S-0-0012.

Structural element	Type	Description
ManufacturerSpecificWarn-ing	BIT	Manufacturer-specific warning, without/with system switch-off, see S-0-0181
Reserved_14	BIT	
Reserved_13	BIT	
CommunicationWarning	BIT	Warning: communication, without/with system switch-off
ExcessiveVelocityDeviation	BIT	Warning: velocity exceedance, without/with system switch-off, see S-0-0377
Reserved_10	BIT	
UndervoltageWarning	BIT	Warning: undervoltage, without/with system switch-off
Reserved_8	BIT	
Reserved_7	BIT	
Reserved_6	BIT	
Reserved_5	BIT	
Reserved_4	BIT	
CoolingErrorWarning	BIT	Warning: cooling, without/with system switch-off
MotorOverTemperature-Warning	BIT	Warning: motor overtemperature, without/with system switch-off, see S-0-0312

Structural element	Type	Description
AmplifierOverTemperature-Warning	BIT	Warning: amplifier overtemperature, without/with system switch-off, see S-0-0311
OverloadWarning	BIT	Warning: overload of amplifier, motor, current supply, without/with system switch-off, see S-0-0310

Tab. 11-93: Elements of the IL_SIII_DRV_WARNING_BITS structure

11.5.27 IL_SIII_DRV_MESSAGE

Brief description The superimposition type (UNION) IL_SIII_DRV_MESSAGE supports access to and display of message bits acc. to S-0-0013.

Structural element	Data type	Description
MessageBits	IL_SIII_DRV_MESSAGE_BITS	Corresponds to parameter S-0-0013 IL_SIII_DRV_MESSAGE_BITS, page 396
MessageWord	WORD	

Tab. 11-94: Elements of the superimposition type IL_SIII_DRV_MESSAGE

11.5.28 IL_SIII_DRV_MESSAGE_BITS

Brief description The IL_SIII_DRV_MESSAGE_BITS structure contains the individual bits of the parameter S-0-0013.

Structural element	Type	Description
ManufacturerSpecificOperationStatus	BIT	See S-0-0182
PositionFeedbackValueStatusSee	BIT	See S-0-0403, BIT 0, "Status of actual position values"
InterpolatorHalted	BIT	See S-0-0343, "Message: Interpolator stopped"
TargetPositionAttained	BIT	See S-0-0342, "Target position reached"
InCoarsePosition	BIT	See S-0-0341, "Message: Coarse In-Position"
MaximumSpindleSpeedExceede	BIT	See S-0-0340, nfeedback >= maximum spindle speed
MinimumSpindleSpeedUndershot	BIT	See S-0-0339, nfeedback <= minimum spindle speed
TargetPositionReached	BIT	See S-0-0338, "Message: In target position" Position feedback = Active target position S-0-0430 – (S-0-0051 OR S-0-0053) < S-0-0057
PowerSupplyExceeded	BIT	See S-0-0337, "P >= Px" Amount of actual power value exceeded settable threshold value
InPosition	BIT	See S-0-0336, "Message: In-Position"
VelocityLimitExceeded	BIT	See S-0-0335, "Message: n_command > n_actual" Command velocity value higher than settable velocity limit value
TorqueLimitExceeded	BIT	See S-0-0334, "Message Md >= Mdlimit" Actual torque or force value reached limit value

Structural element	Type	Description
TorqueThresholdExceeded	BIT	See S-0-0333, "Message Md >= Mdx" Actual torque or force value exceeded settable threshold value
VelocityThresholdUndershot	BIT	See S-0-0332, "Message n_actual < nx" Actual velocity value fallen below a settable threshold value
VelocityInStandstillWindow	BIT	S-0-0331, "Message n_actual = 0" Actual velocity value fallen below a settable threshold value
VelocityInCommandWindow	BIT	S-0-0330, "Message n_actual = n_command" Actual velocity value reached command value within a tolerance window

Tab. 11-95: Elements of the IL_SIII_DRV_MESSAGE_BITS structure

11.5.29 IL_SIII_IO_DIAG

Brief description The IL_SIII_IO_DIAG structure display I/O diagnostic information.

Structural element	Data type	Description
StatusCodeTyp	IL_SIII_IO_DIAG_STATUS_CODE_TYPE	Provides origin of "status code". IL_SIII_IO_DIAG_STATUS_CODE_TYPE, page 397
DiagClass	IL_SIII_DIAGNOSTIC_NUMBER_CLASS	Priority of the present diagnostics IL_SIII_DIAGNOSTIC_NUMBER_CLASS, page 393
StatusCode	IL_SIII_IO_DIAG_STATUS_CODE	"Status code" specified in I/O profile IL_SIII_IO_DIAG_STATUS_CODE, page 398
IoFunctiongroup	UINT	I/O function group causing the diagnostic message e.g. "1503 - Digital output", S-0-1500.0.32, bits 39-24
IoSlotnumber	BYTE	I/O Slot number of the module that the function group is assigned to (structural instance). S-0-1500.0.32, bits 23-16
IoSubbusSlaveIndex	BYTE	Subbus slave triggering the diagnostic message S-0-1500.0.32, bits 15-8
IoChannelNumber	BYTE	Channel triggering the diagnostic message S-0-1500.0.32, bits 7-0
IoOutputsActive	BOOL	Outputs active S-0-1500.0.02, BIT 15
IoInputsValid	BOOL	Inputs valid S-0-1500.0.02, BIT 14

Tab. 11-96: Elements of the IL_SIII_IO_DIAG structure

11.5.30 IL_SIII_IO_DIAG_STATUS_CODE_TYPE

Brief description The enumeration type IL_SIII_IO_DIAG_STATUS_CODE_TYPE provides the "StatusCode" origin.

Element	Value	Description
IO_PROFILE_STATUS_CODE	0	StatusCode defined in I/O profile
MANUFACTURER_STATUS_CODE	1	StatusCode is manufacturer-specific

Tab. 11-97: Elements of the enumeration type IL_SIII_IO_DIAG_STATUS_CODE_TYPE

11.5.31 IL_SIII_IO_DIAG_STATUS_CODE

Brief description The enumeration type IL_SIII_IO_DIAG_STATUS_CODE indicates the "status code" specified in the I/O profile.

The Sercos I/O status code can be found in:

- S-0-0390, Diagnostic message number

  Source type (bits 29-24) | class (bits 19-16) | status code (bits 15-0)

  e.g. 16#C10E2350 ⇒ C1 | 0E | 2350
- S-0-1500.0.32 I/O diagnostic message

  Class (bits 59-56) | status code (bits 55-40) | function group (bits 39-24) | slot number (bits 23-16) | sub-bus slave (bits 15-8) | channel number (bits 7-0)

  e.g. 16#0E235005DE010008 ⇒ 0E | 2350 | 05DE | 01 | 00 | 08

Value	Description
Status codes with the diagnostic classes Operating state (0xC10A), Warning (0xC10E), Error (0xC10F)	
16#0	No error
16#1000	General error
16#2000	Current
16#2100	Current on the side of the device input
16#2110	Short-circuit/earth fault
16#2120	Earth fault
16#2130	Short circuit
16#2136	Short circuit at VCC
16#2137	Short circuit to earth
16#2140	Short circuit in the periphery
16#2141	Cross-fault output and other voltages
16#2142	Cross-fault output and a clock line
16#2146	Short circuit, input at VCC
16#2147	Short circuit, input at earth
16#2150	Open signal lines, not connected
16#2160	Wire break, signal wire
16#2180	Input inactive, not enabled
16#2181	Input circuit faulty (e.g. load)
16#2200	Current within the device
16#2213	Overcurrent upon startup

Value	Description
Status codes with the diagnostic classes Operating state (0xC10A), Warning (0xC10E), Error (0xC10F)	
16#2214	Overcurrent at runtime
16#2220	Permanent overcurrent
16#2230	Short circuit/earth fault
16#2240	Earth fault
16#2250	Short circuit
16#2300	Current on the side of the device output
16#2310	Permanent overcurrent
16#2320	Short-circuit/earth fault
16#2330	Earth fault
16#2340	Short circuit
16#2344	Output overloaded
16#2345	Sensor supply overload
16#2346	Short circuit at VCC
16#2347	Short circuit to earth
16#2350	Wire break
16#2360	Wire break
16#2370	Cross-fault
16#2371	Cross-fault output
16#2372	Cross-fault output
16#2380	Output not enabled
16#2381	Output circuit faulty (e.g. load)
16#3000	Voltage
16#3100	Mains voltage (voltage at the side of the device input - voltage supply)
16#3110	Mains overvoltage
16#3120	Mains undervoltage
16#3130	Phase error
16#3134	Phase sequence
16#3140	Mains frequency
16#3141	Mains frequency too high
16#3142	Mains frequency too low
16#3150	Polarity of supply voltage reverse
16#3180	Signal input, general
16#3181	Signal line open
16#3182	Wire break
16#3183	Unexpected change in signal

Value	Description
Status codes with the diagnostic classes Operating state (0xC10A), Warning (0xC10E), Error (0xC10F)	
16#3184	Overrange, voltage
16#3185	Signal voltage beyond the defined range
16#3186	Underrange, voltage
16#3200	Voltage within the device
16#3210	Overvoltage within the device
16#3220	Undervoltage within the device
16#3230	Loading error
16#3300	Output voltage (voltage at the side of the device output, e.g. supply and operating voltage for external components)
16#3310	Output overvoltage
16#3320	Output undervoltage
16#3400	Supply voltage
16#3401	Supply voltage overvoltage
16#3402	Power supply missing
16#3403	Supply voltage undervoltage
16#3410	Sensor supply
16#3411	Sensor supply, undervoltage
16#3412	Sensor supply missing
16#3413	Sensor supply, overvoltage
16#3420	Actuator supply
16#3421	Actuator supply, undervoltage
16#3422	Actuator supply missing
16#3423	Actuator supply, overvoltage
16#4000	Temperature
16#4100	Ambient temperature
16#4110	Ambient overtemperature
16#4120	Ambient undertemperature
16#4130	Temperature intake air
16#4140	Temperature suction air
16#4200	Device temperature
16#4210	Device overtemperature
16#4220	Device undertemperature
16#4300	External temperature (e.g. of the drive)
16#4310	External overtemperature (e.g. of the drive)
16#4320	External undertemperature (e.g. of the drive)

Value	Description
Status codes with the diagnostic classes Operating state (0xC10A), Warning (0xC10E), Error (0xC10F)	
16#4400	Connection temperature
16#4410	Connection overtemperature
16#4420	Connection undertemperature
16#5000	Device hardware (only in the device)
16#5010	Component defect
16#5100	Supply
16#5110	Low-voltage supply (for all internal voltages 24V, 5V, 3,3V, 3.0V, 2.5V, 1.8V)
16#5120	Air supply
16#5130	Color supply
16#5140	Intermediate circuit supply
16#5150	Initiator supply (fault in supplying an external sensor supply from within the device)
16#5151	Short circuit
16#5160	Supply of the peripheral devices (supply fault for external devices supplied from within the device)
16#5200	Control unit/measuring circuit
16#5210	Measuring circuit
16#5220	Computer circuitries
16#5230	Communication (within the device)
16#5300	Operating unit
16#5400	Power section
16#5410	Output states
16#5420	Chopper (interrupter)
16#5430	Input stages
16#5440	Contactor
16#5450	Fuses
16#5500	Communication with extension modules
16#5600	Internal memory
16#5610	RAM
16#5620	EPROM
16#5630	EEPROM
16#5640	Flash memory
16#6000	Device software
16#6010	Software reset (watchdog)
16#6100	Internal software (firmware)
16#6110	Flexware missing
16#6120	Flexware inconsistent

Value	Description
Status codes with the diagnostic classes Operating state (0xC10A), Warning (0xC10E), Error (0xC10F)	
16#6130	Application not yet ready, booting active
16#6200	Application software
16#6300	Data set incorrect
16#6310	Missing parameter
16#6320	Parameter error
16#6330	Parameter not yet initialized
16#6800	Configuration
16#6810	Process data, mapped multiple times
16#7000	Extension module(s) defective
16#7100	Performance
16#7140	Relay, contactor fault
16#7150	Fuse error
16#7200	Measuring circuit
16#7300	Sensor
16#7400	Computer circuitry
16#7500	Communication
16#7600	Data memory
16#7610	RAM
16#7620	EPROM
16#7630	EEPROM
16#7640	Flash memory
16#7700	Line break/cable fault
16#7710	Wire break between cable and sensor
16#8000	Monitoring
16#8100	Communication
16#8110	Process data monitoring
16#8120	Host monitoring
16#8121	Process data channel, handshake TimeOut
16#8150	Buffer processing
16#8151	Sending buffer
16#8152	Sending buffer, full
16#8153	Sending buffer, overflow
16#8159	Receiving buffer
16#815A	Receiving buffer, full
16#815B	Receiving buffer, overflow

Value	Description
Status codes with the diagnostic classes Operating state (0xC10A), Warning (0xC10E), Error (0xC10F)	
16#8200	Closed-loop control
16#8210	High system deviation (command > actual, manufacturer-specific), (actual deviation more than permitted)
16#8211	Maximum control value reached
16#8220	System deviation (command > actual, manufacturer-specific), (actual deviation more than permitted)
16#8221	Maximum control value reached
16#8900	Sensors
16#8910	Measured value above the value range
16#8920	Measured value below the value range
16#8930	Additional connection for external sensor
16#8A00	Actuator
16#8B00	Preventative maintenance required (state monitoring)
16#8F00	Safety error
16#8F01	Symmetry violation
16#8F08	Light test, output cannot be switched if inactive
16#8F09	Dark test, output cannot be switched off if active
16#8F0F	Error state, module not running anymore
16#9000	External error
16#A000	Subbus slave
16#A001	Subbus slave missing
16#A002	Wrong sub-bus slave present
16#A003	Compatible substitute (expected sub-bus slave replaced by compatible one)
16#A004	Error in the number of sub-bus devices (too many devices)
16#A010	Device error
16#A012	Application on sub-bus slave not ready
16#A013	Subbus device reset
16#A014	Subbus device parameter error
16#A020	Communication error
16#A021	Subbus error - Timeout (break)
16#A022	Multiple transmission error on sub-bus
16#A023	Subbus I/O data communication error
16#A024	Data communication fault in sub-bus management
16#A030	Subbus configuration error
16#A031	I/O data configuration mapped twice
16#A040	General error

Value	Description
Status codes with the diagnostic classes Operating state (0xC10A), Warning (0xC10E), Error (0xC10F)	
16#A041	Subbus hardware error
16#A042	Subbus firmware error
16#A043	Subbus runs asynchronously (with regard to the Sercos bus cycle)
16#B000	Local bus (system bus of the modular station)
16#B001	Local bus slave missing
16#B002	Wrong local bus connected
16#B003	Compatible substitute (expected local bus slave replaced by compatible one)
16#B004	Error in the number of local bus devices (too many devices)
16#B010	Device error
16#B012	Application on local bus slave not ready
16#B013	Local bus device reset
16#B014	Local bus device parameterization missing or faulty
16#B020	Communication error
16#B021	Local bus error - Timeout (interruption)
16#B022	Multiple transmission error on local bus
16#B023	Local bus I/O data communication error
16#B024	Data communication fault in local bus management
16#B025	Ring topology interrupted
16#B030	Configuration error on the local bus
16#B031	I/O data configuration mapped twice
16#B032	Local bus asynchronous to parent bus
16#B040	General error
16#B041	Hardware error on the local bus
16#B042	Firmware error on the local bus
16#B043	Local bus runs asynchronously (with regard to the Sercos bus cycle)
16#F000	Additional functions

Value	Description
Status codes with diagnosis class "procedure command specific state" (0xC10C)	
16#A100	S-0-1508.x.67 Sub-bus Tunnel Command
16#A101	S-0-1508.x.67 Sub-bus Tunnel Command failed. Unsupported Sub-bus Tunnel PDU version
16#A102	S-0-1508.x.67 Sub-bus Tunnel Command failed. Unsupported Sub-bus SDU Type
16#A103	S-0-1508.x.67 Sub-bus Tunnel Command failed. Unsupported Sub-bus Tunnel SDU version
16#A104	S-0-1508.x.67 Sub-bus Tunnel Command failed. Sub-bus slave index wrong
16#A105	S-0-1508.x.67 Sub-bus Tunnel Command failed. Wrong length of SDU

Value	Description
Status codes with diagnosis class "procedure command specific state" (0xC10C)	
16#A106	S-0-1508.x.67 Sub-bus Tunnel Command failed. Wrong length of Command SDU
16#A107	S-0-1508.x.67 Sub-bus Tunnel Command failed. Answer PDU too large
16#A108	S-0-1508.x.67 Sub-bus Tunnel Command failed. Buffer overflow
16#A120	The procedure command S-0-1508.x.80 Sub-bus reconfiguration procedure command is in process
16#A200	The procedure command S-0-1500.x.21 Local bus reconfiguration procedure command is in process
16#BC00	S-0-1500.x.12 Rearrangement of IO resource
16#BC01	Forbidden insertion/replacement - An attempt was made to insert/replace a generic IO function group with an invalid/forbidden slot number
16#BC02	Forbidden insertion/replacement - An attempt was made to insert/replace the function group IO bus coupler (S-0-1500.x.0)
16#BC03	Forbidden insertion/replacement - An attempt was made to insert/replace an undefined generic IO function group
16#BC04	Forbidden insertion/replacement - The list of inserted/replaced function groups (S-0-1500.x.4/11) contains duplicate entries
16#BC05	Forbidden insertion/replacement - The list of inserted/replaced function groups (S-0-1500.x.4/11) is not sorted in an increasing order (slot number, data block number)
16#BC0F	Forbidden insertion/replacement - Manufacturer-specific reason

11.5.32 IL_FBUS_COMMUNICATION_STATE

Brief description The enumeration type **IL_FBUS_COMMUNICATION_STATE** is used to display the status of the field bus communication.

Element	Value	Description
COMM_STATE_NOT_CONFIGURED	16#00000001	Communication is not configured
COMM_STATE_STOP	16#00000002	Communication is in "Stop" state
COMM_STATE_IDLE	16#00000003	Communication is in "Idle" state
COMM_STATE_OPERATE	16#00000004	Communication in cyclic operation

Tab. 11-98: IL_FBUS_COMMUNICATION_STATE enumeration type

11.6 Auxiliary functions (utilities)

11.6.1 IL_SIIIAttribute

Brief description The IL_SIIIAttribute function breaks down the elements of the Sercos attribute.

Target system	Library/folder
MLC	RIL_SercosIII/ Utilities
MTX	RIL_SercosIII/ Utilities
MLD (IL 2G)	RIL_SercosIII/ Utilities

Tab. 11-99: Reference table

Interface description

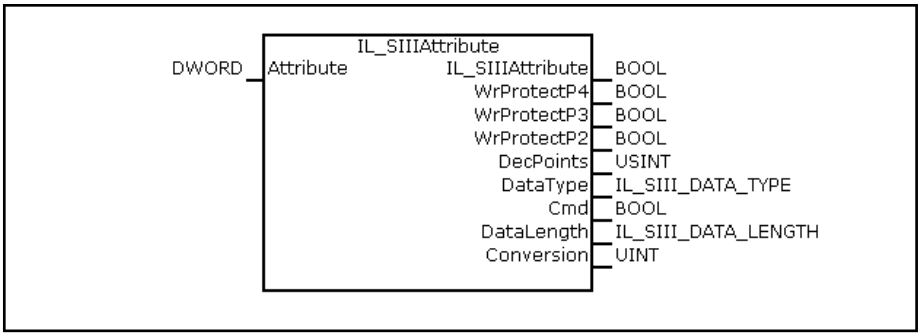


Fig. 11-22: IL_SIIIAttribute function

	Name	Data type	Description
VAR_INPUT	Attributes	DWORD	Sercos attribute
VAR_OUPUT	IL_SIIIAttribute	BOOL	Always TRUE
	WrProtectP4	BOOL	Operating date of the parameter in the operation mode (Sercos phase 4 - P4) write-protected
	WrProtectP3	BOOL	Operating date of the parameter in Sercos phase 3 (P3) write-protected
	WrProtectP2	BOOL	Operating date of the parameter in the parameterization mode (Sercos phase 2 - P2) write-protected
	DecPoints	USINT	Number of decimal places of the operating date of the parameter
	DataType	IL_SIII_DATA_TYPE	Data type and display format of the operating date of the parameter IL_SIII_DATA_TYPE, page 414
	Cmd	BOOL	Operating date calls for command function
	DataLength	IL_SIII_DATA_LENGTH	Data length of the operating date of the parameter IL_SIII_DATA_LENGTH, page 413,
	Conversion	UINT	Evaluation factor of the operating date of the parameter

Tab. 11-100: IL_SIIIAttribute interface

Sercos attribute The following table shows the structure of the Sercos attribute.

Bit	Interface variable	Description
30:	WrProtectP4	Write-protection of the parameter operating date in the operating mode (Sercos phase 4)
		0: Operating date can be written 1: Operating date is write-protected
29:	WrProtectP3	Write-protection of the operating date of the parameter (Sercos phase 3)
		0: Operating date can be written 1: Operating date is write-protected
28:	WrProtectP2	Write-protection of the parameter operating date in the parameterization mode (Sercos phase 2)
		0: Operating date can be written 1: Operating date is write-protected

Bit	Interface variable	Description	
27..24:	DecPoints	The decimal places define the position of the comma for the display and input of the respective operating date.	
		0000 = No decimal places 1111: 15 decimal places (maximum)	
23:		Reserved	
22..20:	DataType	Data type and display format are used to convert the operating date as well as the minimum and maximum input value to the correct display format.	
		Data type	**Display format**
		000: Binary number	Binary
		001: Unsigned integer	Decimal places without signs
		010: Integer	Decimal places with signs
		011: Unsigned integer	hexadecimal
		100: Extended character set	Text
		101: Unsigned integer	Parameter ident number format (Sercos standard, 2 byte)
		110: Floating point number	Decimal places with signs
		111: Unsigned integer	Parameter ident number format (Sercos standard, 4 bytes)
19:	Cmd	The function of the operating date indicates that a command function is called for in the drive with this operating date.	
		0: Operating date or parameter 1: Command	
18..16:	DataLength	The data length is required by the control for the regular completion of the data transmission via the service channel.	
		000: Reserved 001: Operating date is 2 byte long 010: Operating date is 4 byte long 011: Reserved 100: Variable length with 1 byte data 101: Variable length with 2-byte data 110: Variable length with 4-byte data 111: Reserved	
15..0:	Conversion	The evaluation factor is an unsigned integer which serves for the conversion of numeric data to a display format. The evaluation factor is set to the value 1 if it is not used for the display of the dates.	

Tab. 11-101: Elements of the Sercos attribute

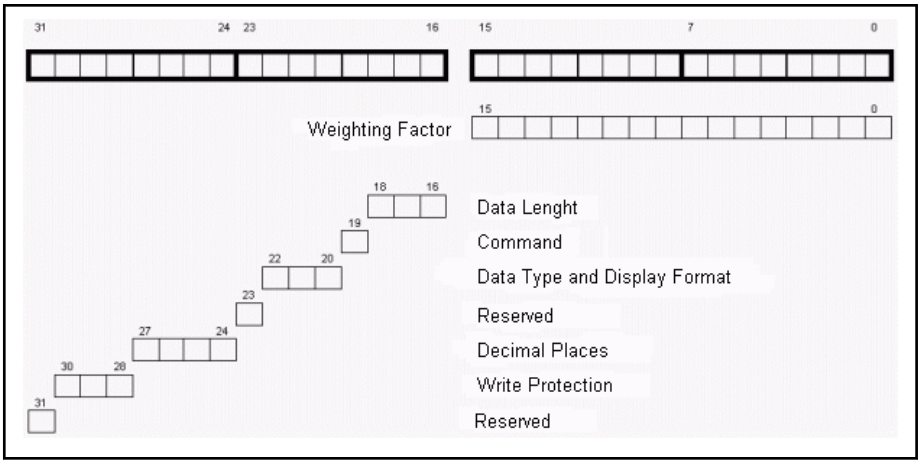


Fig. 11-23: Structure of the Sercos attribute

Functional description

The IL_SIIIAttribute function interprets the Sercos attribute set at the "Attribute" input. The validity of the elements is signaled to the respective outputs by the "IL_SIIIAttribute" output.

Troubleshooting

No troubleshooting available for this function.

11.6.2 IL_SIIIElementsToIdn

Brief description

The function block IL_SIIIElementsToIdn combines the individual elements of an IDN in an MB_IDN value.

Target system	Library/folder
MLC	RIL_SercosIII/ Utilities
MTX	RIL_SercosIII/ Utilities
MLD (IL 2G)	RIL_SercosIII/ Utilities

Tab. 11-102: Reference table

Interface description

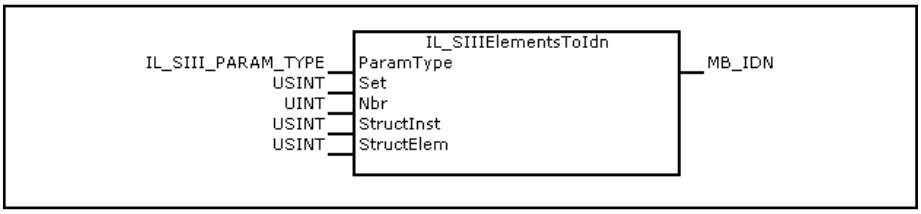


Fig. 11-24: IL_SIIIElementsToIdn function

I/O type	Name	Data type	Description
VAR_INPUT	ParamType	IL_SIII_PARAM_TYPE	Standard (S) or product-specific (P) IDN IL_SIII_PARAM_TYPE, page 414
	Set	USINT	Parameter set
	Nbr	UINT	Data block number
	StructInst	USINT	Structure instance
	StructElem	USINT	Structural element

I/O type	Name	Data type	Description
Return value		MB_IDN	Sercos IDN If the input parameter is invalid, IL_SIIIElementsToIdn = 0

Tab. 11-103: IL_SIIIElementsToIdn interface

11.6.3 IL_SIIIGetSercosPhase

Brief description Use the function IL_SIIIGetSercosPhase to read the Sercos phase of a Sercos bus master.

Target system	Library/folder
MLC	RIL_SercosIII/Utilities
MTX	RIL_SercosIII/Utilities
MLD (IL 2G)	RIL_SercosIII/Utilities

Tab. 11-104: Reference table

Interface description

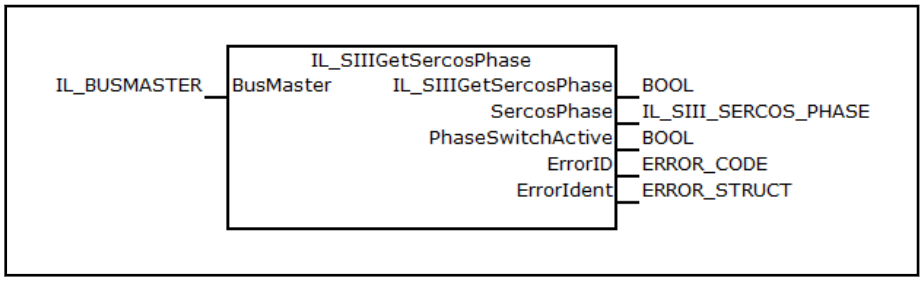


Fig. 11-25: IL_SIIIGetSercosPhase function

I/O type	Name	Data type	Description
VAR_INPUT	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. IL_BUSMASTER, page 415
VAR_OUTPUT	IL_SIIIGetSercosPhase	BOOL	Return value of the function: TRUE: Error occurred, FALSE: Error-free processing
	SercosPhase	IL_SIII_SERCOS_PHASE	Active Sercos phase IL_SIII_SERCOS_PHASE, page 415
	PhaseSwitchActive	BOOL	TRUE: Sercos phase switching is currently active, FALSE: No Sercos phase switching active.
	ErrorID	ERROR_CODE	Diagnostic description in case of error.
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics

Tab. 11-105: IL_SIIIGetSercosPhase interface

Name	Type	Min. value	Max. value	Default value	Effective
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute". Depending on the target system, not all values might be permitted.

Tab. 11-106: Min./max. and default values of the IL_SIIIGetSercosPhase inputs

Functional description

Depending on the target system, multiple Sercos masters can be available. For the assignment of the respective master to the values from "IL_BUSMAS-TER", refer to the documentation of the respective target system. If there is only one Sercos master in the target system, its value is "IL_BUSMAS-TER_0".

The function block tries to read the Sercos phase of the respective bus master. If the Sercos master has not yet been completely initialized or if the phase cannot be read, an error is generated. The value "IL_SIII_PHASE_IN-IT" is returned for the phase and FALSE for the output "PhaseSwitchActive".

The output "PhaseSwitchActive" indicates whether a phase switching is currently active. This output is set to active during the complete phase switching. If it is for example switched from Sercos phase 2 to Sercos phase 4, this output is already active in Sercos phase 2, but also in Sercos phase 3 and until Sercos phase 4 is reached.

If the Sercos phase is continually read using this function, phase switching can be observed. Only after the output "PhaseSwitchActive" becomes FALSE, the final phase is reached.

Troubleshooting

The function block generates errors from the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170) :

ErrorID	Additional1	Additional2	
IN-PUT_RANGE_ERROR	16#00000002	16#00000001	The value specified for the “BusMaster” input is invalid.
STATE_MA-CHINE_ERROR	16#00000006	16#xxxxxxxx	An internal error occurred. Restart the control. If it occurs multiple times, contact the service and report the Additional2 error code for further diagnostics.
SYSTEM_ERROR	16#00000016	16#xxxxxxxx	An internal error occurred. Restart the control. If it occurs multiple times, contact the service and report the Additional2 error code for further diagnostics.
RESOURCE_ERROR	16#00002912	16#00000000	The function is not supported on this target system.
RESOURCE_ERROR	16#00002915	16#00000000	The functionality is not yet available on the control, e.g. the Sercos bus is not yet initialized.

Tab. 11-107: Error codes of the function IL_SIIIGetSercosPhase in F_RELA-TED_TABLE := 16#0170

11.6.4 IL_SIIISetSercosPhase

Brief description

The function block “IL_SIIISetSercosPhase” can be used to set the Sercos phase of a Sercos bus master.

Target system	Library/folder
MLC	RIL_SercosIII/Utilities
MTX	RIL_SercosIII/Utilities
MLD (IL 2G)	RIL_SercosIII/Utilities

Tab. 11-108: Reference table

Interface description

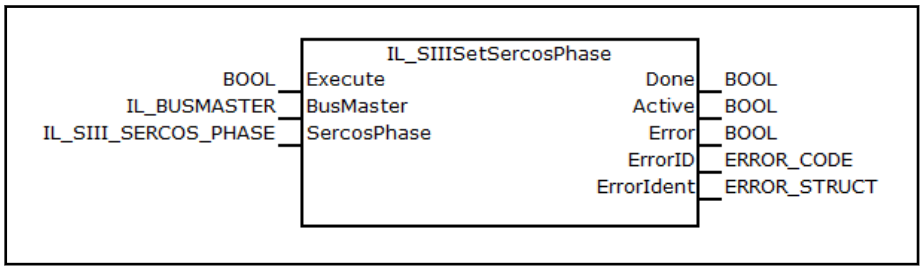


Fig. 11-26: IL_SIIISetSercosPhase function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0. IL_BUSMASTER, page 415
	SercosPhase	IL_SIII_SERCOS_PHASE	Sercos phase to be switched to IL_SIII_SERCOS_PHASE, page 415
VAR_OUTPUT	Done	BOOL	Processing completed without errors, output data valid
	Active	BOOL	Processing active, output data invalid
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics

Tab. 11-109: IL_SIIISetSercosPhase interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4	IL_BUSMASTER_0	Rising edge at "Execute”. Depending on the target system, not all values might be permitted
SercosPhase	IL_SIII_SERCOS_PHASE	IL_SIII_PHASE_0	IL_SIII_PHASE_NRT	IL_SIII_PHASE_4	Rising edge at "Execute”. Depending on the target system, not all values might be permitted

Tab. 11-110: Min./max. and default values of the IL_SIIISetSercosPhase inputs

Troubleshooting

The function block generates errors from the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170) :

ErrorID	Additional1	Additional2	
INPUT_RANGE_ERROR	16#00000002	16#00000001	The value specified for the “BusMaster” input is invalid
STATE_MACHINE_ERROR	16#00000006	16#xxxxxxxx	An internal error occurred. Restart the control. If it occurs multiple times, contact the service and report the Additional2 error code for further diagnostics.
SYSTEM_ERROR	16#00000016	16#00002914	An internal error occurred. Restart the control. If it occurs multiple times, contact the service
ACCESS_ERROR	16#00002910	16#00000000	The phase switching is already active. Another phase switching cannot be commanded
INPUT_RANGE_ERROR	16#00002911	16#00000000	The value specified for the “SercosPhase” input is invalid or the system does not support this phase
RESOURCE_ERROR	16#00002912	16#00000000	The function block is not supported on this target system
RESOURCE_ERROR	16#00002915	16#00000000	The functionality is not yet available on the control, e.g. the Sercos bus is not yet initialized
RESOURCE_ERROR	16#00002916	16#00000000	Phase switching is completed, but the requested phase was not reached. The reason can be an error during phase switching
ACCESS_ERROR	16#00002917	16#00000000	Phase switching can currently not be executed. This can for example result if the drives still have power

Tab. 11-111: Error codes of the IL_SIIISetSercosPhase function block in F_RELATED_TABLE := 16#0170

Functional description

The value "IL_SIII_PHASE_INIT" is not permitted at the "SercosPhase" input. The error code 16#00002911 is then returned. This error can also occur if the target system does not support the requested phase.

Depending on the target system, multiple Sercos masters can be available. For the assignment of the respective master to the values from "IL_BUSMASTER", refer to the documentation of the respective target system. If there is only one Sercos master in the target system, its value is "IL_BUSMASTER_0".

The function block tries to switch the Sercos phase of the respective bus master. If the execution is successful, the "Done" output is set. Otherwise, the "Error" output is set. The period until one of the two outputs is set varies strongly due to its dependency on the number of devices to be connected at the respective bus. If time monitoring is required, it has to be implemented by the user.

11.6.5 IL_SIIIStringToIdn

Brief description

The function IL_SIIIStringToIdn converts a string such as "S-0-0001.0.0" or the shortened versions "S-0-0001" or "S1" to an MB_IDN value.

Target system	Library/folder
MLC	RIL_SercosIII/ Utilities
MTX	RIL_SercosIII/ Utilities
MLD (IL 2G)	RIL_SercosIII/ Utilities

Tab. 11-112: Reference table

Interface description

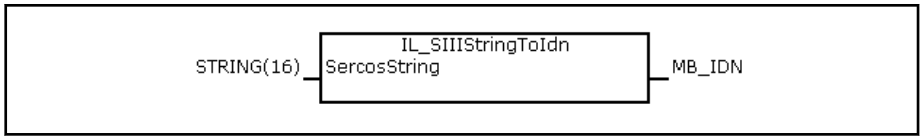


Fig. 11-27: IL_SIIIStringToIdn function

I/O type	Name	Data type	Description
VAR_INPUT	SercosString	STRING(16)	Parameter string "S-0-0001.0.0" or shortened versions "S-0-0001" or "S1"
Return value		MB_IDN	Sercos IDN If the input parameter is invalid, IL_SIIIStringToIdn = -1

Tab. 11-113: IL_SIIIStringToIdn interface

Functional description

The function IL_SIIIStringToIdn converts a Sercos parameter string with the following strings elements (formatting: "X-Y-ZZZZ.SI.SE" or "X-Y-ZZZZ" or "XZ"):

- SercosType "X": "S", "P", "s", "p" (for MLC also "A", "C", "K" "M", "N", "O", "a", "c", "k", "m", "n", "o")
- SercosSet "Y": "0"..."7" (optional only for "XZ")
- SercosNo "ZZZZ": "0"..."4095" (only for "X-Y-ZZZZ" and "XZ" possible without leading zeroes)
- Structure instance "SI": "0"..."255" (optional, together with SE)
- Structural element "SE": "0"..."255" (optional, together with SI)
- Separator: "-" or "."

11.6.6 IL_SIII_DATA_LENGTH

Brief description

The enumeration type supports the analysis of the Sercos attributes with regard to data length and differentiation between list and individual parameters.

Library	Folder
RIL_SercosIII	Utilities

Tab. 11-114: Library assignment

Program:

```
TYPE IL_SIII_DATA_LENGTH :
(
FIX_2_BYTE := 16#01, (* Operation data is two octets long *)
FIX_4_BYTE := 16#02, (* Operation data is four octets long *)
FIX_8_BYTE := 16#03, (* Operation data is eight octets long *)
VAR_1_BYTE := 16#04, (* Variable length with one-octet data strings *)
VAR_2_BYTE := 16#05, (* Variable length with two-octet data strings *)
VAR_4_BYTE := 16#06, (* Variable length with four-octet data strings *)
VAR_8_BYTE := 16#07 (* Variable length with eight-octet data strings *)
)USINT;
END_TYPE
```

Functional description

The enumeration type supports the analysis of the Sercos attributes with regard to data length (_x_Byte -> x=data length) and differentiation between list (VAR_x_BYTE) and individual parameters (FIX_x_BYTE).

The data type is used at the "DataLength" output of the function IL_SIIIAttribute, page 405,.

11.6.7 IL_SIII_DATA_TYPE

Brief description The enumeration type supports the analysis of the Sercos attributes with regard to data type and display format.

Library	Folder
RIL_SercosIII	Utilities

Tab. 11-115: Library assignment

Program:

```
TYPE IL_SIII_DATA_TYPE :
(
    BINARY  := 16#00, (* binary number *)
    UDEC    := 16#01, (* unsigned integer, display Decimal *)
    DEC     := 16#02, (* signed integer, display Decimal *)
    HEX     := 16#03, (* unsigned integer, display Hexadecimal *)
    TEXT    := 16#04, (* extended character set *)
    IDN     := 16#05, (* unsigned integer, display IDN "S-0-0000.0.0" *)
    FLOAT   := 16#06, (* floating-point number *)
    STIME   := 16#07  (* Sercos time *)
)USINT;
END_TYPE
```

Functional description The enumeration type supports the analysis of the Sercos attributes with regard to data type and display format.

The data type is used at the "DataType" output of the function IL_SIIIAttribute, page 405,.

11.6.8 IL_SIII_ELEMENT

Brief description

The IL_SIII_ELEMENT enumeration type describes the Sercos elements of a parameter.

Library	Folder
RIL_SercosIII	Utilities

Tab. 11-116: Library assignment

Name	Value	Description
IL_STATUS	1	Service channel "Data Status"
IL_NAME	2	Name
IL_ATTRIBUTE	3	Attributes
IL_UNIT	4	Unit
IL_MINVALUE	5	Minimal Value
IL_MAXVALUE	6	Maximal Value
IL_OPDATA	7	Date

Tab. 11-117: Elements of enumeration type IL_SIII_ELEMENT

11.6.9 IL_SIII_PARAM_TYPE

Brief description The IL_SIII_PARAM_TYPE enumeration type distinguishes between standard IDN and product-specific IDN.

Library	Folder
RIL_SercosIII	Utilities

Tab. 11-118: Library assignment

Name	Value	Description
IL_S_PARAM	0	Standard IDN
IL_P_PARAM	1	Product-specific IDN

Tab. 11-119: Elements of enumeration type IL_SIII_PARAM_TYPE

11.6.10 IL_SIII_SERCOS_PHASE

Type definitions

The following definition describes the enumeration type IL_SIII_SERCOS_PHASE.

The type definitions are included in the library. The value “Wert IL_SIII_PHASE_INIT” is always specified if the Sercos phase cannot be determined, e.g. if the Sercos bus has not yet been initialized or if the phase cannot be read due to an error.

The data type "IL_SIII_SERCOS_PHASE" is defined as follows:

Enumeration element	Value	Description
IL_SIII_PHASE_0	0	Sercos phase 0
IL_SIII_PHASE_1	1	Sercos phase 1
IL_SIII_PHASE_2	2	Sercos phase 2
IL_SIII_PHASE_3	3	Sercos phase 3
IL_SIII_PHASE_4	4	Sercos phase 4
IL_SIII_PHASE_NRT	7	No Sercos phase active, IP communication possible
IL_SIII_PHASE_INIT	16#FFFFFFFF	Invalid Sercos phase or phase cannot be determined

Tab. 11-120: Defining the enumeration elements of the IL_SIII_SERCOS_PHASE data type

11.6.11 IL_BUSMASTER

Brief description

The IL_BUSMASTER enumeration type of the RIL_FieldbusTypes.library is used to select the appropriate Sercos III master.

Currently, only the onboard master is supported.

Element	Value	Description
IL_BUSMASTER_0	0	Onboard master
IL_BUSMASTER_1	1	Not supported
IL_BUSMASTER_2	2	Not supported
IL_BUSMASTER_3	3	Not supported
IL_BUSMASTER_4	4	Not supported

Tab. 11-121: IL_BUSMASTER enumeration type

12 RMB_SercosIII_Util.library

12.1 Introduction and overview

The RMB_SercosIII_Util.library includes function blocks to control Sercos III devices.

Target systems

The library can be used in the following systems:

Target system	Library
MLC/MTX	RMB_SercosIII_Util
MLD with IndraLogic 2G for 20VRS and above	RMB_SercosIII_Util

Drive as PLC device

In the systems MLC and MLD, the drive axes are controlled via function blocks acc. to "PLCopen".

In addition to these function blocks, there is the "AxisInterface". The AxisInterface pools and extends PLCopen motion function blocks and provides an easy-to-operate interface for the drive functionality.

The number of axes is limited and the motion axes request the resources of the control.

The following function blocks "PlcDrive" are used for a simplified axis control via the Sercos bus. The drive is created as PLC device (also refer to Drive as PLC device).

No motion axis instance is assigned and the required resources of the control are minimal. The operation modes of the drive (interpolation in the drive) are used. The requirements on the respective drive are minimal. Third-party drives (e.g. without scaling system) can be used via these function blocks.

Restrictions when using these function blocks:

- For these axes, the use of only **one** operation mode is intended. It has to be set via the engineering tool of the device.
- To configure the cyclic communication and diagnostics of the PLC devices, IndraWorks dialogs are provided. However, the device cannot be commissioned directly on the device node in IndraWorks. For Rexroth devices, IndraWorks Ds can be used via a direct connection (refer to the help Drive as PLC device). For third-party tools, use the respective tool.



For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).



An "AxisInterface for Sercos PLC devices" example program is available as GAT plugin "SercosPlcDriveInterface". This example program can also be used without GAT. Refer to the plugin documentation.

The example program comprises the following function blocks and operation modes:

- MB_SIIIPlcDriveVelocityType01 - Velocity control
- MB_SIIIPlcDriveTorqueType01 - Torque control
- MB_SIIIPlcDrivePosInterpolationType01 - Drive-internal interpolation (only in version 1 of the plugin)
- MB_SIIIPlcDrivePositioningType01 - Drive-controlled positioning (as of version 2 of the plugin)
- MB_SIIIPlcDriveEfcFrequencyType01 - Frequency control

Overview on the library content

The "Drives" folder includes function blocks for the basic Sercos operation modes "Velocity control", "Torque control", "Drive-internal interpolation" and "Drive-controlled positioning".

"PowerSupply" folder: Function blocks to control supply units with Sercos.

"RexrothEFC" folder: A function block for the operation mode "Frequency control" and auxiliary functions for the Rexroth frequency converter EFC with Sercos.

The "RexrothHydraulicDrives" folder includes function blocks for the hydraulic modes "Flow control", "Pressure/flow control", "pressure/swivel angle control", "Direct valve control", "Torque/direct valve control", "Torque/flow control", "Pressure control/direct valve control" and "velocity control with alternating torque control".

"RexrothSytronix" folder: Function blocks to control the Sytronix devices DRn 5020 and FcP5020 with Sercos.

Areas of application and function blocks of RMB_SercosIII_Util

Area of application (folder in library)	
Function blocks	**Description**
Drives (function blocks for a simplified axis control)	
MB_SIIIPlcDriveType01, page 422	Basic function block extended by the other function blocks.
MB_SIIIPlcDriveVelocityType01, page 426	Extended MB_SIIIPlcDriveType01. The function block is implemented for the "velocity control" mode
MB_SIIIPlcDriveTorqueType01, page 430	Extended MB_SIIIPlcDriveType01. The function block is implemented for the "torque control" mode
MB_SIIIPlcDrivePositioningType01, page 434	Extended MB_SIIIPlcDriveType01. The function block is implemented for the "drive-controlled positioning" mode
MB_SIIIPlcDrivePosInterpolationType01, page 440	Extended MB_SIIIPlcDriveType01. The function block is implemented for the "drive-internal interpolation" mode
With data types:	
MB_SIII_CONTROLWORD, page 444	Corresponds to the parameter “Master control word” S-0-0134
MB_SIII_STATUSWORD, page 445	Corresponds to the parameter “Master status word” S-0-0135

MB_SIII_DRIVE_SHUTDOWN_MODE, page 445	Allows to select the response when switching off drive enabled
MB_SIII_HOMING_MODE, page 445	Allows to select the response when establishing a position data reference
MB_SIII_POSCTRLWORD, page 446	Corresponds to the parameter “Positioning control word” S-0-0346
MB_SIII_IN_PROC_VELOCITY, page 447	Contains process input data of the function block MB_SIIIPlcDriveVelocityType01
MB_SIII_OUT_PROC_VELOCITY, page 447	Contains process output data of the function block MB_SIIIPlcDriveVelocityType01
MB_SIII_IN_PROC_TORQUE, page 447	Contains process input data of the function block MB_SIIIPlcDriveTorqueType01
MB_SIII_OUT_PROC_TORQUE, page 447	Contains process output data of the function block MB_SIIIPlcDriveTorqueType01
MB_SIII_IN_PROC_POSITIONING, page 448	Contains process input data of the function block MB_SIIIPlcDrivePositioningType01
MB_SIII_OUT_PROC_POSITIONING, page 448	Contains process output data of the function block MB_SIIIPlcDrivePositioningType01
MB_SIII_IN_PROC_POS_INTERPOLATION, page 448	Contains process input data of the function block MB_SIIIPlcDrivePosInterpolationType01
MB_SIII_OUT_PROC_POS_INTERPOLATION, page 448	Contains process output data of the function block MB_SIIIPlcDrivePosInterpolationType01
PowerSupply	
MB_SIIIPowerSupplyDriveIntegrated, page 450	Can identify the power supply state, handle the transitions between the power supply state machines, bring the power supply into the required state and control the operation mode.
MB_SIIIPowerSupplyStandalone, page 458	The function is equal to MB_SIIIPowerSupplyDriveIntegrated. The function block can also reset a power supply error and provide diagnostic information on the state of the power supply
MB_SIIIPowerSupplyDcDcIslandGrid, page 465	The function is equal to MB_SIIIPowerSupplyStandalone for supply units for the extended supply unit functions DC-DC converter or microgrid mode.
With data types:	
MB_POWERSUPPLY_ACTUAL_MODE, page 472	Enumeration of the states of the state machine of a power supply
MB_POWERSUPPLY_ERROR_MODE, page 473	Enumeration, to which state the power supply switches when exiting an internal error state
MB_POWERSUPPLY_MODE, page 472	Enumeration of the operation modes of the power supply
MB_POWERSUPPLY_TARGET_MODE, page 474	Enumeration of the available target modes for the function blocks MB_SIIIPowerSupplyDriveIntegrated and MB_SIIIPowerSupplyStandalone. The target mode decides on the final state to be reached by the power supply as well as on the path to be taken to reach the final state.

MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID, page 475	Enumeration of the available target modes for the function block MB_SIIIPowerSupplyDcDcIslandGrid. The target mode decides on the final state to be reached by the power supply as well as on the path to be taken to reach the final state.
MB_POWERSUPPLY_FUNCTION, page 475	Enumeration of the extended supply unit functions for the function block MB_SIIIPowerSupplyDcDcIslandGrid.
RexrothEFC	
MB_SIIIEfcCodeToIdn, page 476	Converts a function code/parameter code of the EFC into the corresponding Sercos parameter IDN.
MB_SIIIPlcDriveEfcFrequencyType01, page 477	Used to operate a Rexroth frequency converter EFC in the mode "Frequency control".
With data types:	
MB_EFC_FUNCTION_CODE, page 481	Describes the function code of the Rexroth frequency converter EFC
MB_SIII_IN_PROC_EFC_FREQ, page 481	Contains process input data of the function block MB_SIIIPlcDriveEfcFrequencyType01
MB_SIII_OUT_PROC_EFC_FREQ, page 481	Contains process output data of the function block MB_SIIIPlcDriveEfcFrequencyType01
MB_SIII_EFC_CONTROLWORD, page 482	Corresponds to the parameter P-0-1098, register H0.00 "Register for the communication control" of the EFC
MB_SIII_EFC_STATUSWORD, page 482	Corresponds to the parameter P-0-1098.0.1, register H0.01 "Register for feedback of communication status" of the EFC
RexrothHydraulicDrives	
MB_SIIIPlcDrivePressureFlowControl, page 487	Implements the "Pressure and flow control" mode for Rexroth HydraulicDrives
MB_SIIIPlcDrivePressureSwivelAngleControl, page 492	Implements the "Pressure and swivel angle control" mode for Rexroth HydraulicDrives
MB_SIIIPlcDrivePressureSwivelAngleSlave, page 497	Implements the "Pressure and swivel angle control (for slave)" mode for Rexroth HydraulicDrives
MB_SIIIPlcDriveDirectValveControl, page 501	Implements the "Direct valve control" mode for Rexroth HydraulicDrives
MB_SIIIPlcDriveTorqueDirectValveControl, page 505	Implements the "Direct valve control" mode for Rexroth HydraulicDrives
MB_SIIIPlcDriveTorqueFlowControl, page 511	Implements the "Torque/force/direct valve control" mode for Rexroth HydraulicDrives
MB_SIIIPlcDrivePressureDirectValveControl, page 516	Implements the "Pressure control/Direct valve control" mode for Rexroth HydraulicDrives
MB_SIIIPlcDriveVelocityAltCtrl, page 521	Implements the "Velocity control with alternating force control" mode for Rexroth HydraulicDrives
With data types:	
MB_SIII_HYDR_CONTROL_MODE, page 526	Enumeration type, contains the possible operation modes of Rexroth HydraulicDrives
MB_SIII_IN_PROC_FLOW, page 526	Input structure for the input "InProc" of the function block MB_SIIIPlcDriveFlowControl

MB_SIII_OUT_PROC_FLOW, page 527	Output structure for the output "OutProc" of the function block MB_SIIIPlcDriveFlowControl
MB_SIII_IN_PROC_PRESSURE_FLOW, page 527	Input structure for the input "InProc" of the function block MB_SIIIPlcDrivePressureFlowControl
MB_SIII_OUT_PROC_PRESSURE_FLOW, page 528	Output structure for the output "OutProc" of the function block MB_SIIIPlcDrivePressureFlowControl
MB_SIII_IN_PROC_SWIVEL, page 529	Input structure for the input "InProc" of the function block MB_SIIIPlcDrivePressureSwivelAngleControl
MB_SIII_OUT_PROC_SWIVEL, page 529	Output structure for the output "OutProc" of the function block MB_SIIIPlcDrivePressureSwivelAngleControl
MB_SIII_IN_PROC_SWIVEL_SLAVE, page 530	Input structure for the input "InProc" of the function block MB_SIIIPlcDrivePressureSwivelAngleSlave
MB_SIII_OUT_PROC_SWIVEL_SLAVE, page 530	Output structure for the output "OutProc" of the function block MB_SIIIPlcDrivePressureSwivelAngleSlave
MB_SIII_IN_PROC_SPOOL, page 531	Input structure for the input "InProc" of the function block MB_SIIIPlcDriveDirectValveControl
MB_SIII_OUT_PROC_SPOOL, page 531	Output structure for the output "OutProc" of the function block MB_SIIIPlcDriveDirectValveControl
MB_SIII_IN_PROC_TORQUE_SPOOL, page 532	Input structure for the input "InProc" of the function block MB_SIIIPlcDriveTorqueDirectValveControl
MB_SIII_OUT_PROC_TORQUE_SPOOL, page 532	Output structure for the output "OutProc" of the function block MB_SIIIPlcDriveTorqueDirectValveControl
MB_SIII_IN_PROC_TORQUE_FLOW, page 532	Input structure for the input "InProc" of the function block MB_SIIIPlcDriveTorqueFlowControl
MB_SIII_OUT_PROC_TORQUE_FLOW, page 533	Output structure for the output "OutProc" of the function block MB_SIIIPlcDriveTorqueFlowControl
MB_SIII_IN_PROC_PRESSURE_SPOOL, page 533	Input structure for the input "InProc" of the function block MB_SIIIPlcDrivePressureDirectValveContro
MB_SIII_OUT_PROC_PRESSURE_SPOOL, page 534	Output structure for the output "OutProc" of the function block MB_SIIIPlcDrivePressureDirectValveControl
MB_SIII_IN_PROC_VEL_ALT_CTRL, page 534	Input structure for the input "InProc" of the function block MB_SIIIPlcDriveVelocityAltCtrl
MB_SIII_OUT_PROC_VEL_ALT_CTRL, page 535	Output structure for the output "OutProc" of the function block MB_SIIIPlcDriveVelocityAltCtrl
RexrothSytronix	
MB_SIIIDRn5020, page 535	Allows the pressure and swivel angle control of the EFC5610 based on Sytronix DRn5020 via Sercos communication
MB_SIIIFcP5020, page 541	Allows the pressure and flow control of the EFC 5610 based on Sytronix FcP 5020 via Sercos communication
With data types:	
MB_SIII_DRN_ENABLE, page 545	The enumeration is used by the DRnEnable input of the function block MB_SIII_DRn5020
MB_SIII_DRN_VAR_VEL_ENABLE, page 545	The enumeration is used by the DRnVarVelocityEnable input of the function block MB_SIII_DRn5020

MB_SIII_FCP_MASTERSLAVE, page 545	The enumeration is used by the MasterSlave input of the MB_SIII_FcP5020 function block
MB_SIII_FCP_PARAGROUP, page 546	The enumeration is used by the pQParameterGroup input and by the pQParameterGroupStatus output of the MB_SIII_FcP5020 function block
MB_SIII_FCP_DUALPUMPSTATUS, page 546	The enumeration is used by the TwoPointDualPumpStatus output of the MB_SIII_FcP5020 function block

Tab. 12-1: RMB_SercosIII_Util library

12.2 Drives

12.2.1 MB_SIIIPlcDriveType01

Brief description

The MB_SIIIPlcDriveType01 function block is used as basis for the following function blocks assigned to a specified operation mode.

The main operation mode is always enabled.



The function block "MB_SIIIPlcDrive" was renamed to " MB_SIIIPlcDriveType01" in 14V16 and the interface was optimized.

Target system	Library
MLC/MTX for 14V10 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for 20VRS and higher	RMB_SercosIII_Util

Tab. 12-2: Reference table

Interface description

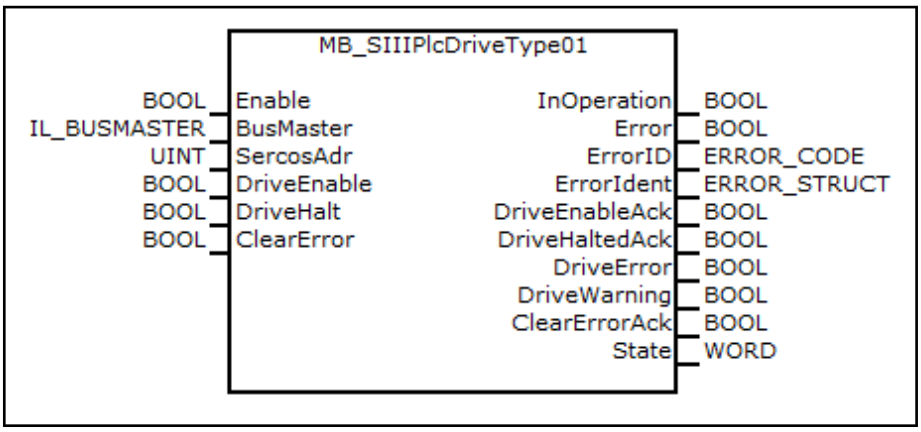


Fig. 12-1: MB_SIIIPlcDriveType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device.

I/O type	Name	Data type	Description
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
VAR_OUT-PUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 426
	DriveEnableAck	BOOL	Drive enabled acknowledged, S-0-0135, Drive status word, bit 14/15
	DriveHaltedAck	BOOL	Drive halt acknowledged, S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error, S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning, S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed, S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Current drive status. See below.
PROPERTY			
IN	DriveShutdown-ModeProp	MB_SIII_DRIVE_SHUT-DOWN_MODE	Selection to switch off drive enabled at a negative edge at "DriveEnable". Options: Deceleration or torque enabled. MB_SIII_DRIVE_SHUTDOWN_MODE on page 445
IN	StatusWord	WORD	Assign input variable %IW of the status word S-0-0135 to the "wControlword" element.
OUT	ControlWord	WORD	Assign output variable %QW of the control word S-0-0134 to the "wControlword" element.

Tab. 12-3: MB_SIIIPlcDriveType01 interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available

Bit no.	Name	Description
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Bus master	IL_BUSMASTER			0	Rising edge at “Enable”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	System-specific	0	Continuous
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
DriveShutdown-ModeProp	MB_SIII_DRIVE_SHUT-DOWN_MODE			DRIVE_SHUT-DOWN_TOR-QUE_MAX	Writing the property
StatusWord	WORD			0	Writing the property

Tab. 12-4: Min./max. and default values of the inputs MB_SIIIPlcDriveType01

Functional description

The function block interprets the status word S-0-0135 of the drive and generates the control word S-0-0134 which sends the required commands to the drive. These two parameters have to be configured cyclically.

The "ClearError" input enables the command "Clear error" S-0-0099.

The "DriveShutdownModeProp" input is used to define the behavior when resetting the "DriveEnable" input. Controlling the bits 14 and 15 in S-0-0134 is thus affected.



Some inputs and outputs are implemented as "property" to use them more flexibly in the derived function blocks.

The "DriveShutDownMode" input is only implemented into a derived function block if the respective device supports the modes.

Process data (StatusWord and ControlWord) are summarized in structures in the derived function blocks.

Configuring cyclic data in Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

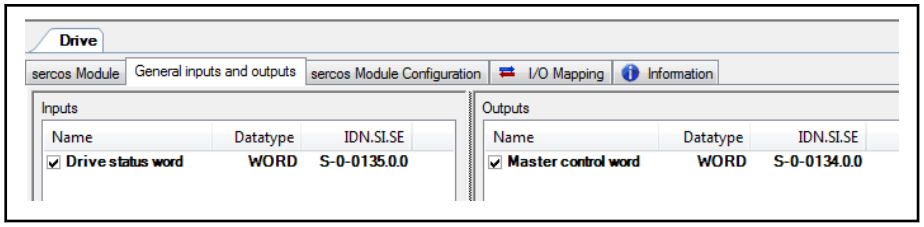


Fig. 12-2: Configuring cyclic parameters

Subsequently, the corresponding input variables and output variables are assigned in the "I/O mapping" tab. As the status word and the control word are implemented as "property", it is required to create intermediate variables and to link them in the I/O image. The status word is copied to the property "Status word" **before** calling the function block. The control word is copied to the property "Control word" **after** calling the function block.

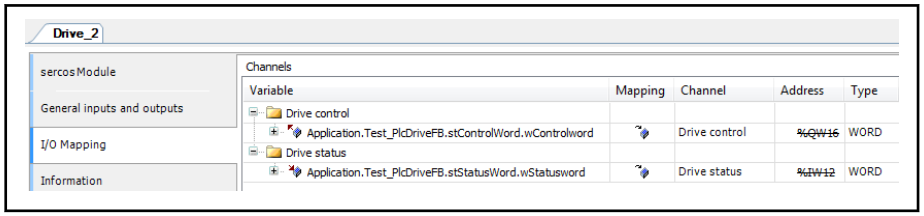


Fig. 12-3: Mapping input variables and output variables

Parameterizing the drive:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The MB_SIIIPlcDriveType01 function block only requires the parameterization of the main operation mode S-0-0032. Depending on the operation mode, further parameters have to be configured.

A basic parameterization of the drive (limit values, etc.) is required in any case.

Using MB_SIIIPlcDriveType01 as basis for the individual function blocks:

To implement individual function blocks for further operation modes, MB_SIIIPlcDriveType01 can be used.

In this example, the optional input "DriveShutdownMode" is implemented as well.

Implement the following two structures:

MY_SIIII_IN_PROC_Xxx

```
TYPE MY_SIII_IN_PROC_Xxx :
STRUCT
  Statusword: MB_SIII_STATUSWORD;         // S-0-0135 status word
  MyActValue: DINT:= 16#0000000000000000; // S-0-xxxx actual value
  // define more inputs here
END_STRUCT
END_TYPE
```

MY_SIIII_OUT_PROC_Xxx

```
TYPE MY_SIII_OUT_PROC_Xxx :
STRUCT
  Controlword: MB_SIII_CONTROLWORD;        // S-0-0134 control word
  MYCmdValue:  DINT:= 16#0000000000000000; // S-0-xxxx setpoint
  // define more outputs here
```

```
END_STRUCT
END_TYPE
```

Implementation of the derived function block:

MY_SIIISpecialPlcDrive (declaration)

```
FUNCTION_BLOCK MY_SIIISpecialPlcDrive EXTENDS MB_SIIIPlcDriveType01
VAR_INPUT
 DriveShutdownMode: MB_SIII_DRIVE_SHUTDOWN_MODE := DRIVE_SHUTDOWN_TORQUE_MAX;
                            // Shut down torque free or immediatelly stop
 MyCmd:     REAL; // setpoint
 InProc:    MY_SIII_IN_PROC_Xxx;  // Input structure
END_VAR
VAR_OUTPUT
 MyAct:     REAL; // actual value
 OutProc:   MY_SIII_OUT_PROC_Xxx; // Output structure
END_VAR
```

MY_SIIISpecialPlcDrive (implementation)

```
// The following inputs and outputs have to be copied to properties of the base FB,
// because we do not want these in- and outputs visible. Statusword and Controlword are here structures.
// DriveShutdownMode is not used in all FBs
SUPER^.StatusWord :=  InProc.Statusword.wStatusword;
SUPER^.DriveShutdownModeProp := DriveShutdownMode;
SUPER^();
OutProc.Controlword.wControlword := SUPER^.ControlWord;
```

Implement the following two methods:

Initialization:

```
METHOD PUBLIC mCalcInit: BOOL
```

`mCalcInit := FALSE;` indicates that the initialization is completed.

Cyclic call:

```
METHOD mCalcInOp
```

Troubleshooting In addition to the errors of the internally used function blocks IL_SIIICommand, IL_SIIIMasterState and IL_SIIIRemoteSlaveState, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000002	SercosAdr	The input value of the parameter "SercosAdr" is outside the valid range
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
RESOURCE_ERROR	16#00002980	16#00000000	Sercos bus is not in the operating mode P4
RESOURCE_ERROR	16#00002981	State	Device failed while "InOperation". See above.

Tab. 12-5: Error codes of the MB_SIIIPlcDriveType01 function block

12.2.2 MB_SIIIPlcDriveVelocityType01

Brief description The MB_SIIIPlcDriveVelocityType01 function block is used to operate a drive in the "velocity control" mode.



The function block "MB_SIIIVelocityPlcDrive" was renamed to " MB_SIIIPlcDriveVelocityType01" in 14V16 and the interface was optimized.



An "AxisInterface" example program is available as GAT plugin "SercosPlcDriveInterface" for this function block. This example program can also be used without GAT. Refer to the plugin documentation.

Target system	Library
MLC/MTX ≥14V10	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ 20VRS	RMB_SercosIII_Util

Tab. 12-6: Reference table

Interface description

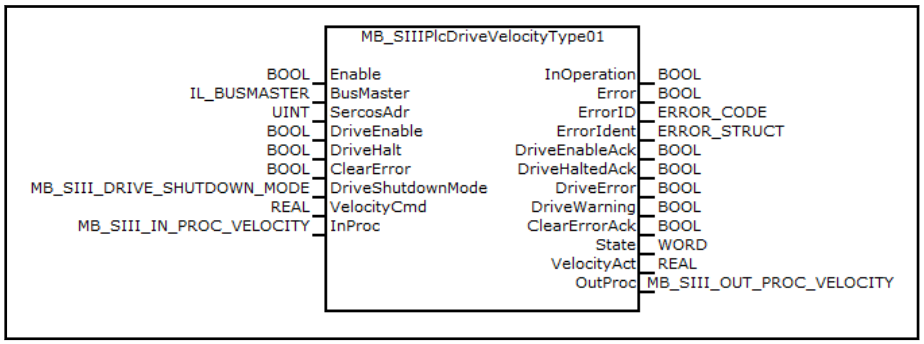


Fig. 12-4: MB_SIIIPlcDriveVelocityType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	DriveShutdown-Mode	MB_SIII_DRIVE_SHUT-DOWN_MODE	Selection to switch off drive enabled at a negative edge at "DriveEnable". Options: Deceleration or torque enabled. MB_SIII_DRIVE_SHUTDOWN_MODE, page 445
	VelocityCmd	REAL	Velocity command value. Unit e.g. in [rpm], according to parameterization.

I/O type	Name	Data type	Description
	InProc	MB_SIII_IN_PROC_VELOCITY	Applying process input data %I. MB_SIII_IN_PROC_VELOCITY, page 447
VAR_OUTPUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 430
	DriveEnableAck	BOOL	Drive enabled acknowledged, S-0-0135, Drive status word, bit 14/15
	DriveHaltedAck	BOOL	Drive halt acknowledged, S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error, S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning, S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed, S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Current drive status.
	VelocityAct	REAL	Actual velocity value. Unit e.g. in [rpm], according to parameterization.
	OutProc	MB_SIII_OUT_PROC_VELOCITY	Apply process output data %Q. MB_SIII_OUT_PROC_VELOCITY, page 447

Tab. 12-7: MB_SIIIPlcDriveVelocityType01 interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Bus master	IL_BUSMASTER			0	Rising edge at “Enable”

Name	Type	Min. value	Max. value	Default value	Effective
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	System-specific	0	Continuous
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
DriveShutdown-Mode	MB_SIII_DRIVE_SHUT-DOWN_MODE			DRIVE_SHUT-DOWN_TOR-QUE_MAX	Continuous
VelocityCmd	REAL			0.0	Continuous
InProc	MB_SIII_IN_PROC_VE-LOCITY			0	Continuous

Tab. 12-8: Min./max. and default values of the inputs MB_SIIIPlcDriveVelocity-Type01

Functional description

The MB_SIIIPlcDriveVelocityType01 function block extends the MB_SIIIPlc-DriveType01 function block and is provided with all its inputs and outputs as well as its functionality.

Additionally, it is provided with inputs and outputs for the actual and command values of this operation mode:

- The "VelocityCmd" input indicates the velocity command value of the axis. The value is cyclically transferred to the drive via the "OutProc.Velocity" output and the parameter S-0-0036.
- The "VelocityAct" output outputs the actual velocity value of the axis. The value is cyclically read by the drive via the parameter S-0-0036 and the input "InProc.Velocity".

Configuring cyclic data in Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

At first, configure the cyclic parameters in the “General inputs and outputs” tab.

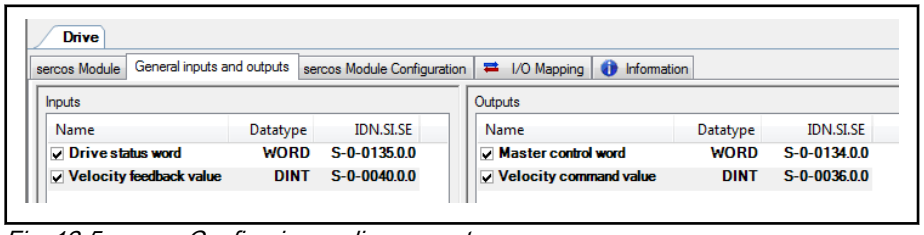


Fig. 12-5: Configuring cyclic parameters

Subsequently, the corresponding input and output variables of the function block are assigned in the "I/O mapping" tab.

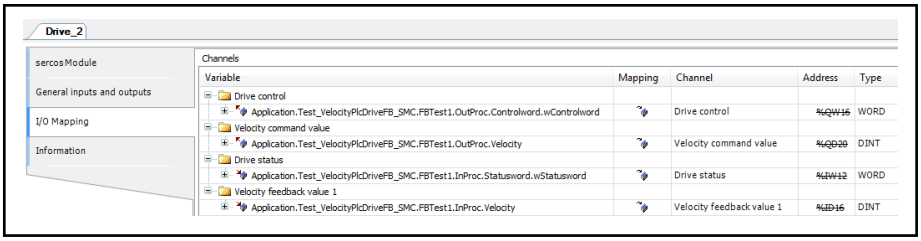


Fig. 12-6: Mapping input variables and output variables

Parameterizing the drive:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The MB_ SIIIVelocityPlcDriveType01 function block requires the parameterization of the main operation mode S-0-0032 to "velocity control" (value 2#0000.0000.0000.0010).

For the further parameterization for the "velocity control" mode (acceleration, etc.), use IndraWorks Ds for example.

A basic parameterization of the drive (limit values, etc.) is required in any case.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002982	Attribute value	Attribute of the parameter S-0-0036 or S-0-0040 is not four bytes decimal with sign.

Tab. 12-9: Error codes of the MB_SIIIPlcDriveVelocityType01 function block

12.2.3 MB_SIIIPlcDriveTorqueType01

Brief description

The MB_SIIIPlcDriveTorqueType01 function block is used to operate a drive in the operation mode "torque control".

The function block "MB_SIIITorquePlcDrive" was renamed to " MB_SIIIPlcDriveTorqueType01" in 14V16 and the interface was optimized.

An "AxisInterface" example program is available as GAT plugin "SercosPlcDriveInterface" for this function block. This example program can also be used without GAT. Refer to the plugin documentation.

Target system	Library
MLC/MTX ≥14V10	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ 20VRS	RMB_SercosIII_Util

Tab. 12-10: Reference table

Interface description

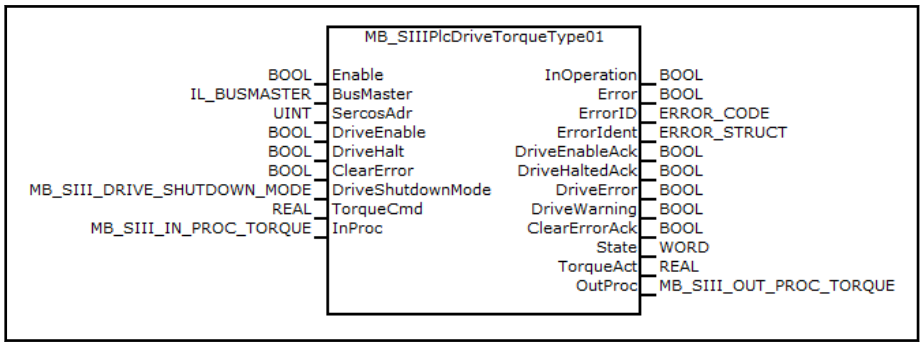


Fig. 12-7: MB_SIIIPlcDriveTorqueType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	Bus master	IL_BUSMAS-TER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`.
	SercosAdr	UINT	Sercos III address of the slave device.
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	DriveShutdown-Mode	MB_SIII_DRIVE_SHUT-DOWN_MODE	Selection to switch off drive enabled at a negative edge at "DriveEnable". Options: Deceleration or torque enabled. MB_SIII_DRIVE_SHUTDOWN_MODE, page 445
	TorqueCmd	REAL	Torque command value. Unit, e.g. in [%], according to parameterization.
	InProc	MB_SIII_IN_PROC_TOR-QUE	Applying process input data %I. MB_SIII_IN_PROC_TORQUE, page 447
VAR_OUT-PUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error

I/O type	Name	Data type	Description
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 434
	DriveEnableAck	BOOL	Drive enabled acknowledged, S-0-0135, Drive status word, bit 14/15
	DriveHaltedAck	BOOL	Drive halt acknowledged, S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error, S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning, S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed, S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Current drive status. See below.
	TorqueAct	REAL	Actual torque value. Unit, e.g. in [%], according to parameterization.
	OutProc	MB_SIII_OUT_PROC_TOR-QUE	Apply process output data %Q. MB_SIII_OUT_PROC_TORQUE, page 447

Tab. 12-11: MB_SIIIPlcDriveTorqueType01 interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Bus master	IL_BUSMASTER			0	Rising edge at “Enable”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	System-specific	0	Continuous
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
DriveShutdown-Mode	MB_SIII_DRIVE_SHUT-DOWN_MODE			DRIVE_SHUT-DOWN_TOR-QUE_MAX	Continuous
TorqueCmd	REAL			0.0	Continuous
InProc	MB_SIII_IN_PROC_TOR-QUE			0	Continuous

Tab. 12-12: Min./max. and default values of the inputs MB_SIIIPlcDriveTorque-Type01

Functional description

The MB_SIIIPlcDriveTorqueType01 function block extends the MB_SIIIPlc-DriveType01 function block and is provided with all its inputs and outputs as well as its functionality.

Additionally, it is provided with inputs and outputs for the actual and command values of this operation mode:

- The "TorqueCmd" input indicates the torque command value of the axis. The value is cyclically transferred to the drive via the "OutProc.Torque" output and the parameter S-0-0080.
- The "TorqueAct" output outputs the actual torque value of the axis. The value is cyclically read by the drive via the parameter S-0-0084 and the input "InProc.Torque".

Configuring cyclic data in Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

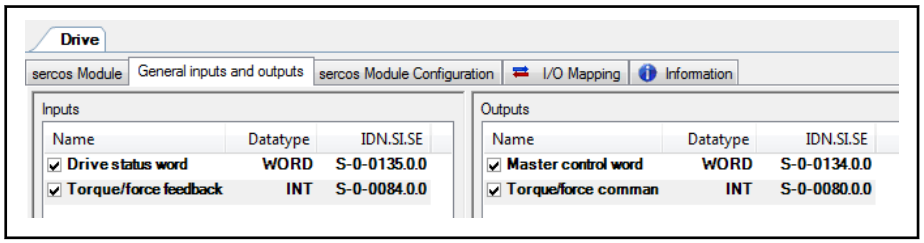


Fig. 12-8: Configuring cyclic parameters

Subsequently, the corresponding input variables and output variables of the function block are assigned in the “I/O mapping” tab.

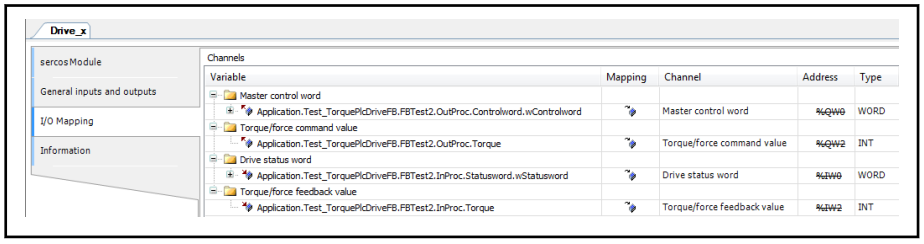


Fig. 12-9: Mapping input variables and output variables

Parameterizing the drive:



For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The MB_SIIIPlcDriveTorqueType01 requires the parameterization of the main operation mode S-0-0032 to "torque control" (value 2#0000.0000.0000.0001).

For the further parameterization for the "torque control" mode (ramp, etc.), use IndraWorks Ds for example.

A basic parameterization of the drive (limit values, etc.) is required in any case.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002984	Attribute value	Attribute of the parameter S-0-0080 or S-0-0084 is not two bytes decimal with sign.

Tab. 12-13: Error codes of the MB_SIIIPlcDriveTorqueType01 function block

12.2.4 MB_SIIIPlcDrivePositioningType01

Brief description

The MB_SIIIPlcDrivePositioningType01 function block is used to operate a drive in the "Drive-controlled positioning" mode.



An "AxisInterface" example program is available as GAT plugin "SercosPlcDriveInterface" for this function block. This example program can also be used without GAT.

Target system	Library
MLC/MTX ≥ 14V22	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ 20VRS	RMB_SercosIII_Util

Tab. 12-14: Reference table

Interface description

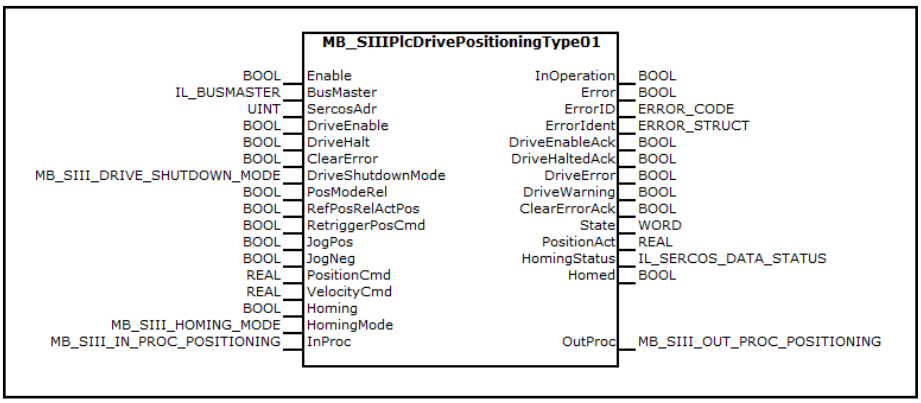


Fig. 12-10: MB_SIIIPlcDrivePositioningType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID = IL_BUSMASTER_0 Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device
	DriveEnable	BOOL	Enabling the drive (AH/AF)
	DriveHalt	BOOL	Stopping the drive (AH)
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command S-0-0099. After the error has been cleared, the selected enabling can be activated again
	DriveShutdown-Mode	MB_SIII_DRIVE_SHUTDOWN_MODE	Selection to switch off drive enabled at a negative edge at "DriveEnable" Options: Deceleration or torque enabled MB_SIII_DRIVE_SHUTDOWN_MODE, page 445
	PosModeRel	BOOL	Type of positioning command value FALSE: Absolute TRUE: Relative
	RefPosRelAct-Pos	BOOL	Reference point for positioning command values FALSE: Latest effective target position (S-0-0430) TRUE: Active actual position value (S-0-0386)
	Retrigger-PosCmd	BOOL	Executing last positioning command again
	JogPos	BOOL	Endless traveling in positive direction
	JogNeg	BOOL	Endless traveling in negative direction
	PositionCmd	REAL	Command position. Unit, e.g. in [degrees], according to parameterization
	VelocityCmd	REAL	Positioning velocity. Unit e.g. in [rpm], according to parameterization
	Homing	BOOL	Establishing position data reference
	HomingMode	MB_SIII_HOMING_MODE	S-0-0148, C0600 Drive-controlled homing procedure command or S-0-0447, C0300 Set absolute position procedure command MB_SIII_HOMING_MODE, page 445
	InProc	MB_SIII_OUT_PROC_POSITIONING	Applying process input data %I
VAR_OUT-PUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error

I/O type	Name	Data type	Description
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics "Troubleshooting" on page 439
	DriveEnableAck	BOOL	Drive enabled acknowledged, from S-0-0135, Drive status word, bits 14/15
	DriveHaltedAck	BOOL	Drive halt acknowledged, from S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error, from S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning, from S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed: S-0-0099, C0500 Reset class 1 diagnostics, Bit 13
	State	WORD	Current drive status. Refer to the “State” output of the IL_SIIIRemoteSlaveState function block
	PositionAct	REAL	Actual position value. Unit, e.g. in [degrees], according to parameterization
	HomingStatus	IL_SERCOS_DATA_STATUS	Status of the homing command IL_SERCOS_DATA_STATUS, page 361
	Homed	BOOL	Drive homed, from status word S-0-0135, Drive status word, bit 4
	OutProc	MB_SIII_IN_PROC_POSITIONING	Applying process output data %Q

Tab. 12-15: MB_SIIIPlcDrivePositioningType01 interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Bus master	IL_BUSMASTER			0	Rising edge at “Enable”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	System-specific	0	Continuous
DriveEnable	BOOL			FALSE	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
DriveShutdown-Mode	MB_SIII_DRIVE_SHUTDOWN_MODE			DRIVE_SHUT-DOWN_TORQUE_MAX	Continuous
PosModeRel	BOOL			FALSE	Continuous
RefPosRelAct-Pos	BOOL			FALSE	Continuous
Retrigger-PosCmd	BOOL			FALSE	Continuous
JogPos	BOOL			FALSE	Continuous
JogNeg	BOOL			FALSE	Continuous
PositionCmd	REAL			0.0	Continuous
VelocityCmd	REAL			0.0	Continuous
Homing	BOOL			FALSE	Continuous
HomingMode	MB_SIII_HOM-ING_MODE			DRIVE_CONTROL-LED_HOMING	Continuous
InProc	MB_SIII_IN_PROC_POSITIONING			0	Continuous

Tab. 12-16: Min./max. and default values of the inputs MB_SIIIPlcDrivePositioningType01

Functional description

The MB_SIIIPlcDrivePositioningType01 function block extends the MB_SIIIPlcDriveType01 function block and is provided with all its inputs and outputs as well as its functionality.

Additionally, it is provided with inputs and outputs for the operation modes “Drive-controlled positioning” and “Jogging”:

- "PositionCmd" input: Position command value of the axis. The value is cyclically transferred to the drive via the "OutProc.Position" output and the parameter S-0-0282, Positioning command value. If this value changes, the positioning command is executed immediately. To again execute a positioning command with the same value, an edge is required at the "RetriggerPosCmd" input.
- "VelocityCmd" input: Positioning velocity of the axis. The value is cyclically transferred to the drive via the "OutProc.Velocity" output and the parameter S-0-0259, Positioning velocity.
- "PosModeRel" input:
  - FALSE = Absolute positioning;
  - TRUE = Relative positioning (reference point for the positioning command value, refer to "RefPosRelActPos")
- "RefPosRelActPos" input:
  - FALSE = Additive. Reference point for the positioning command value; last effective target position S-0-0340
  - TRUE = Relative. Reference point for the positioning command value, active actual position value S-0-0386

The three possible positioning types with the two inputs "PosModeRel" and "RefPosRelActPos" are shown below in the "time diagram Positioning".

- PosModeRel = FALSE. RefPosRelActPos = FALSE: **Absolute** positioning. That means that each positioning command is always traversed with regard to the reference.
- PosModeRel = TRUE. RefPosRelActPos = FALSE: **Additive** positioning. The positioning command is added to the parameter S-0-0430 Effective target position of the drive
- PosModeRel = TRUE. RefPosRelActPos = TRUE: **Relative** positioning. The motion refers to the parameter S-0-0386 Active actual position value

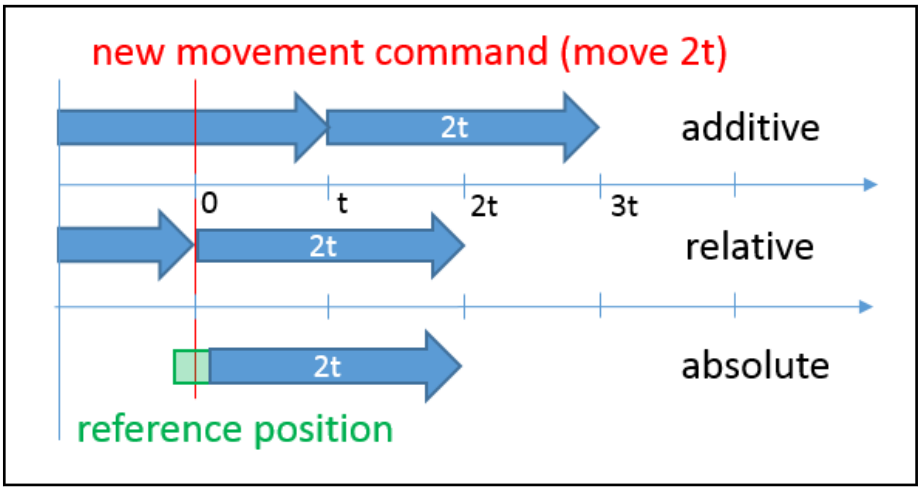


Fig. 12-11: Time diagram - Positioning

- "RetriggerPosCmd" input: FALSE -> TRUE = Last positioning command is executed again
- "JogPos" input: TRUE = Endless traveling in positive direction
- "JogNeg" input: TRUE = Endless traveling in negative direction
- "PositionAct" output: Actual velocity value of the axis. The value is cyclically read by the drive via either the parameter S-0-0386 (Active position feedback value), S-0-0051 (Position feedback value of encoder 1) or S-0-0053 (Position feedback value of encoder 2) and the input "InProc.Position".

There are the following inputs and outputs to establish the position data reference:

- “HomingMode” input: S-0-0148, C0600 Drive-controlled homing procedure command or S-0-0447, C0300 Set absolute position procedure command
- "Homing" input: Starting the command selected via the "HomingMode"
- "HomingStatus" output: Status of the homing command
- "Homed" output: Status of the actual position values, TRUE = Drive in homing

Configuring cyclic data in Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

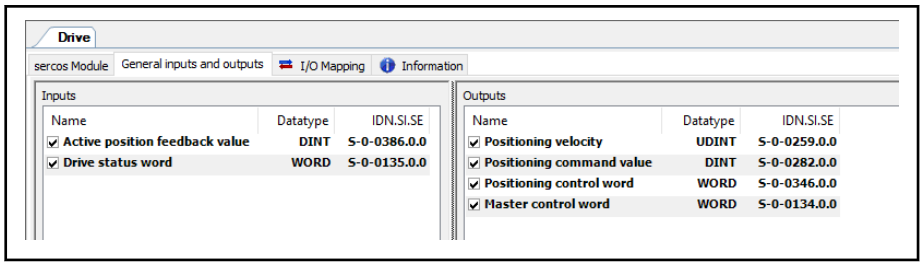


Fig. 12-12: Configuring cyclic parameters

Subsequently, the corresponding input and output variables of the function block are assigned in the "I/O mapping" tab.

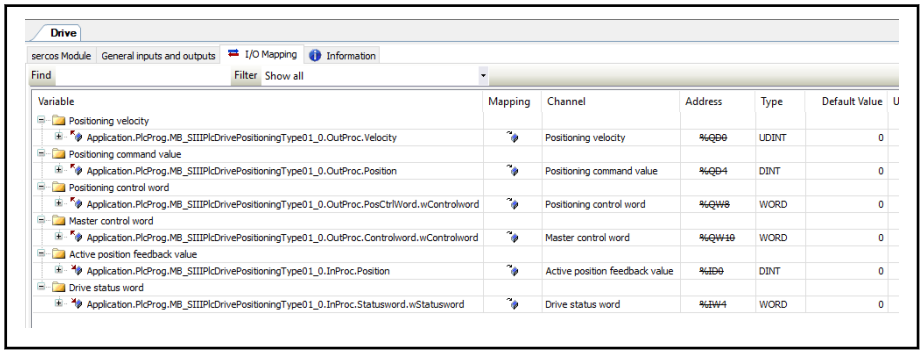


Fig. 12-13: Mapping input variables and output variables

Parameterizing the drive:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The MB_SIIIPositiongPlcDrive function block requires the parameterization of the main operation mode S-0-0032 to "drive-controlled positioning" (value 0b0000.00xx.0010.xxxx).

For the further parameterization for the "drive-controlled positioning" mode (acceleration, etc.), use IndraWorks Ds for example.

A basic parameterization of the drive (limit values, etc.) is required in any case.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Addition-al2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002983	Attribute value	Attribute of the parameters S-0-0386, S-0-0282 or S-0-0259 is not four bytes decimal with sign

Tab. 12-17: Error codes of the MB_SIIIPlcDrivePositioningType01 function block

12.2.5 MB_SIIIPlcDrivePosInterpolationType01

Brief description

The MB_SIIIPlcDrivePosInterpolationType01 function block is used to operate a drive in the "drive-internal interpolation" mode.



The function block "MB_SIIIPositioningPlcDrive" was renamed to " MB_SIIIPlcDrivePosInterpolationType01" in 14V16 and the interface was optimized.



An "AxisInterface" example program is available as GAT plugin "SercosPlcDriveInterface" for this function block. This example program can also be used without GAT. Refer to the plugin documentation.

Target system	Library
MLC/MTX ≥14V10	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ 20VRS	RMB_SercosIII_Util

Tab. 12-18: Reference table

Interface description

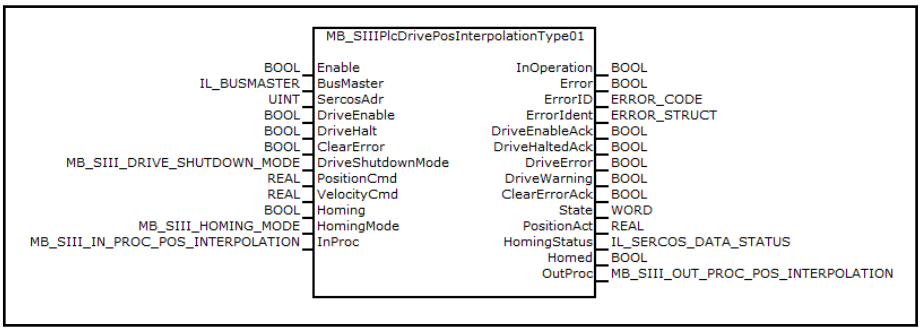


Fig. 12-14: MB_SIIIPlcDrivePosInterpolationType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	DriveEnable	BOOL	Enabling the drive (AH/AF)
	DriveHalt	BOOL	Stopping the drive (AH)
	ClearError	BOOL	Clearing an error in the drive using the “Clear error” command S-0-0099. After clearing, activate the selected enabling.
	DriveShutdown-Mode	MB_SIII_DRIVE_SHUTDOWN_MODE	Selection to switch off drive enabled at a negative edge at "DriveEnable". Options: Deceleration or torque enabled. MB_SIII_DRIVE_SHUTDOWN_MODE, page 445

I/O type	Name	Data type	Description
	PositionCmd	REAL	Command position. Unit, e.g. in [degrees], according to parameterization.
	VelocityCmd	REAL	Positioning velocity. Unit e.g. in [rpm], according to parameterization.
	Homing	BOOL	Establishing position data reference
	HomingMode	MB_SIII_HOMING_MODE	S-0-0148, C0600 Drive-controlled homing procedure command or S-0-0447, C0300 Set absolute position procedure command MB_SIII_HOMING_MODE, page 445
	InProc	MB_SIII_IN_PROC_POS_INTERPOLATION	Applying process input data %I.
VAR_OUTPUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics "Troubleshooting" on page 444
	DriveEnableAck	BOOL	Drive enabled acknowledged, from S-0-0135, Drive status word, bits 14/15.
	DriveHaltedAck	BOOL	Drive halt acknowledged, from S-0-0135, Drive status word, bit 4.
	DriveError	BOOL	Drive error, from S-0-0135, Drive status word, bit 13.
	DriveWarning	BOOL	Drive warning, from S-0-0135, Drive status word, bit 12.
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed: S-0-0099, C0500 Reset class 1 diagnostics, bit 13.
	State	WORD	Current drive status. Refer to the “State” output of the IL_SIIIRemoteSlaveState function block.
	PositionAct	REAL	Actual position value. Unit, e.g. in [degrees], according to parameterization.
	HomingStatus	IL_SERCOS_DATA_STATUS	Status of the homing command IL_SERCOS_DATA_STATUS, page 361
	Homed	BOOL	Drive homed, from status word S-0-0135, Drive status word, bit 4.
	OutProc	MB_SIII_OUT_PROC_POS_INTERPOLATION	Apply process output data %Q.

Tab. 12-19: MB_SIIIPlcDrivePosInterpolationType01 interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "FieldbusStates" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready

Bit no.	Name	Description
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Bus master	IL_BUSMASTER			0	Rising edge at “Enable”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	System-specific	0	Continuous
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
DriveShutdown-Mode	MB_SIII_DRIVE_SHUTDOWN_MODE			DRIVE_SHUT-DOWN_TORQUE_MAX	Continuous
ClearError	BOOL			FALSE	Continuous
PositionCmd	REAL			0.0	Continuous
VelocityCmd	REAL			0.0	Continuous
Homing	BOOL			FALSE	Continuous
HomingMode	MB_SIII_HOM-ING_MODE			DRIVE_CONTROL-LED_HOMING	Continuous
InProc	MB_SIII_IN_PROC_POS_INTERPOLATION			0	Continuous

Tab. 12-20: Min./max. and default values of the inputs MB_SIIIPlcDrivePosInterpolationType01

Functional description

The MB_SIIIPlcDrivePosInterpolationType01 function block extends the MB_SIIIPlcDriveType01 function block and is provided with all its inputs and outputs as well as its functionality.

Additionally, it is provided with inputs and outputs for the actual and command values of this operation mode:

- The "PositionCmd" input indicates the position command value of the axis. The value is cyclically transferred to the drive via the "OutProc.Position" output and the parameter S-0-0258, Target position.
- The "VelocityCmd" input indicates the positioning velocity of the axis. The value is cyclically transferred to the drive via the "OutProc.Velocity" output and the parameter S-0-0259, Positioning velocity.
- The "PositionAct" output outputs the actual velocity value of the axis. The value is cyclically read by the drive via the parameter S-0-0051

(Position feedback value of encoder 1) or S-0-0053 (Position feedback value of encoder 2) and the input "InProc.Position".

There are the following inputs and outputs to establish the position data reference:

- “HomingMode” input: S-0-0148, C0600 Drive-controlled homing procedure command or S-0-0447, C0300 Set absolute position procedure command
- "Homing" input: Starting the command selected via the "HomingMode"
- "HomingStatus" output: Status of the homing command
- "Homed" output: Status of the actual position values, TRUE = Drive in homing

The "drive-internal interpolation" used here only supports the absolute positioning.

For the relative positioning, the command value "PositionCmd" is calculated using the actual value "PositionAct".

For additive positioning, the parameter SS-0-0430, Effective target position, is additionally used cyclically and the command value "PositionCmd" is calculated using S-0-0430.

Configuring cyclic data in Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

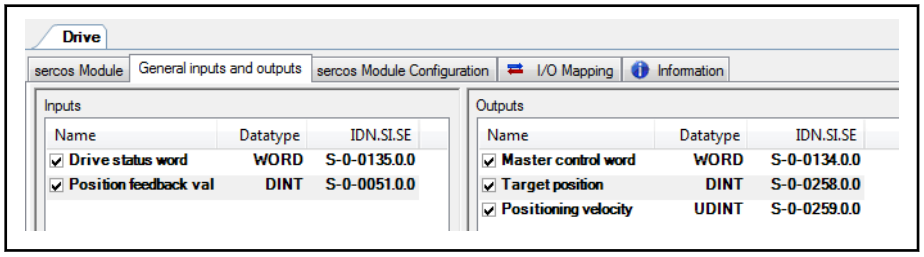


Fig. 12-15: Configuring cyclic parameters

Subsequently, the corresponding input and output variables of the function block are assigned in the "I/O mapping" tab.

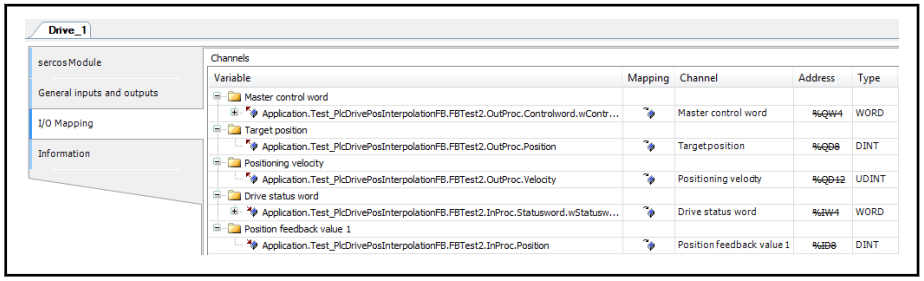


Fig. 12-16: Mapping input variables and output variables

Parameterizing the drive:



For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The MB_SIIIPositiongPlcDrive function block requires the parameterization of the main operation mode S-0-0032 to "drive-internal interpolation" (value 0b0000.00xx.0001.xxxx).

For the further parameterization for the "drive-internal interpolation" mode (acceleration, etc.), use IndraWorks Ds for example.

A basic parameterization of the drive (limit values, etc.) is required in any case.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Addition-al2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002983	Attribute value	Attribute of the parameters S-0-0051, S-0-0258 or S-0-0259 is not four bytes decimal with sign.

Tab. 12-21: Error codes of the MB_SIIIPlcDrivePosInterpolationType01 function block

12.2.6 MB_SIII_CONTROLWORD

Brief description

The data structure MB_SIII_CONTROLWORD corresponds to the parameter “master status word” S-0-0134. The bits can only be read in the user program.

The variable “wControlword“ is stored in the same memory as the bits and is used to map in the I/O mapping of the drive.

Structural element	Type	Min. value	Max. value	Default value	Description
wControlword	WORD			0	Complete check word
OpModeBit8	BIT			FALSE	Bits 8, 9 and 11 indicate the operation mode selected. The main operation mode is always enabled.
OpModeBit9	BIT			FALSE	
OpModeBit11	BIT			FALSE	
Drive_STOP	BIT			FALSE	1->0 drive halt (only if bit 14 and 15 = 1)
Drive_Release	BIT			FALSE	1->0 undelayed torque switch-off
Drive_ON	BIT			FALSE	1->0 best possible standstill (only with bit 14 = 1)

12.2.7 MB_SIII_STATUSWORD

Brief description The data structure MB_SIII_STATUSWORD corresponds to the parameter “master status word” S-0-0135. The bits can only be read in the user program.

The variable “wStatusword“ is stored in the same memory as the bits and is used to map in the I/O mapping of the drive.

Structural element	Type	Min. value	Max. value	Default value	Description
wStatusword	WORD			0	Complete status word
StatusCmdValueProcessing	BIT			FALSE	Status of command value processing
StatusDriveHalt	BIT			FALSE	Status of drive-HALT
Homed	BIT			FALSE	Status of the actual position values
ActOpModeBit8	BIT			FALSE	Actual operation mode 000: Primary operation mode active 001: 1 secondary operation mode active 010: 2 secondary operation mode, etc.
ActOpModeBit9	BIT			FALSE	
ActOpModeBit10	BIT			FALSE	
Warning	BIT			FALSE	Warning ZKL 2
Error	BIT			FALSE	Drive error ZKL 1
EnabledBit14	BIT			FALSE	Readiness: 00: Not ready for power on (e.g. P2); 01: Ready for power on ("bb”); 10: Control and power section ready for operation (Ab); 1: In operation, under torque (e.g. AF)
EnabledBit15	BIT			FALSE	

Tab. 12-22: Elements of the MB_SIII_STATUSWORD structure

12.2.8 MB_SIII_DRIVE_SHUTDOWN_MODE

Brief description The enumeration type MB_SIII_DRIVE_SHUTDOWN_MODE can be used to select the behavior when switching off drive enable.

Element	Value	Description
DRIVE_SHUTDOWN_TORQUE_MAX	0	Deceleration with maximum acceleration
DRIVE_SHUTDOWN_TORQUE_FREE	1	Torque enables (coasting)

Tab. 12-23: Elements of the MB_SIII_DRIVE_SHUTDOWN_MODE enumeration type

12.2.9 MB_SIII_HOMING_MODE

Brief description The enumeration type MB_SIII_HOMING_MODE can be used to select the behavior when establishing a position data reference.

Element	Value	Description
DRIVE_CONTROLLED_HOMING	0148	S-0-0148, C0600 Drive-controlled homing procedure command
SET_ABSOLUTE_POSITION	0447	S-0-0447, C0300 Set absolute position procedure command

Tab. 12-24: Elements of the MB_SIII_HOMING_MODE enumeration type

12.2.10 MB_SIII_POSCTRLWORD

Brief description The data structure MB_SIII_POSCTRLWORD corresponds to the parameter “Master control word” S-0-0346. The bits can only be read in the user program.

The variable "wControlword" is stored in the same memory as the bits and is used to map in the I/O mapping of the drive.

Structural element	Type	Min. value	Max. value	Default value	Description
wControlword	WORD			0	Complete check word
PosPointAcception	BIT			FALSE	Applying position command value with toggling
ActivatePos1	BIT			FALSE	00: Enabling the position Positioning active, start by toggling bit 0 Positioning abort by: 01: Endless traveling in positive direction (Jog+) 10: Endless traveling in negative direction (Jog-) 11: Stopping the axis (positioning halt)
ActivatePos2	BIT			FALSE	
PosTypeRel	BIT			FALSE	Type of positioning command value 0 - Absolute 1: Relative (depends on bit 4)
RefPointActPos	BIT			FALSE	Reference point for positioning command values 0: Latest effective target position (S-0-0430) 1: Active actual position value (S-0-0386)
ImmediateBlock-Change	BIT			FALSE	Immediate block switching 0: The current target position is approached before positioning on the new target position 1: Immediate block switching, i.e. the new target position is approached immediately
SeqBlockBehavior1	BIT			FALSE	Note: These bits are not used by the function block MB_SIIIPlcDrivePositioning! Response if subsequent block (bit 5 = 0) 00: Stopping at target position of starting block 01: Crossing target position of starting block (mode 1) 10: Crossing target position of starting block (mode 2)
SeqBlockBehavior2	BIT			FALSE	

12.2.11 MB_SIII_IN_PROC_VELOCITY

Brief description The data structure MB_SIII_IN_PROC_VELOCITY contains the process input data of the function block MB_SIIIPlcDriveVelocityType01 on page 426.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135 to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
Velocity	DINT	16#0	Assign input variable %ID of the actual velocity value S-0-0040.

Tab. 12-25: Elements of the MB_SIII_IN_PROC_VELOCITY structure

12.2.12 MB_SIII_OUT_PROC_VELOCITY

Brief description The data structure MB_SIII_OUT_PROC_VELOCITY contains the process output data of the function block MB_SIIIPlcDriveVelocityType01 on page 426.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134 to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
Velocity	DINT	16#0	Assign output variable %QD of the velocity command value S-0-0036.

Tab. 12-26: Elements of the MB_SIII_OUT_PROC_VELOCITY structure

12.2.13 MB_SIII_IN_PROC_TORQUE

Brief description The data structure MB_SIII_IN_PROC_TORQUE contains the process input data of the function block MB_SIIIPlcDriveTorqueType01 on page 430.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135 to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
Torque	INT	16#0	Assign input variables %IW of the actual torque value S-0-0084.

Tab. 12-27: Elements of the MB_SIII_IN_PROC_TORQUE structure

12.2.14 MB_SIII_OUT_PROC_TORQUE

Brief description The data structure MB_SIII_OUT_PROC_TORQUE contains the process output data of the function block MB_SIIIPlcDriveTorqueType01 on page 430.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134 to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
Torque	INT	16#0	Assign output variable %QW of the torque command value S-0-0080.

Tab. 12-28: Elements of the MB_SIII_OUT_PROC_TORQUE structure

12.2.15 MB_SIII_IN_PROC_POSITIONING

Brief description The data structure MB_SIII_IN_PROC_POSITIONING contains the process input data of the function block MB_SIIIPlcDrivePositioningType01 on page 434.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135 to the "wStatusword" element MB_SIII_STATUSWORD on page 445
Position	DINT	16#0	Assign input variable %IW of the current position S-0-0386 or position S-0-0051

Tab. 12-29: Elements of the MB_SIII_IN_PROC_POSITIONING structure

12.2.16 MB_SIII_OUT_PROC_POSITIONING

Brief description The data structure MB_SIII_OUT_PROC_POSITIONING contains the process output data of the function block MB_SIIIPlcDrivePosInterpolationType01 on page 440.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134 to the "wControlword" element MB_SIII_CONTROLWORD on page 444
PosCtrlWord	MB_SIII_POSCTRLWORD		Assign output variable %QW of the control word S-0-0346 to the "wControlword" element MB_SIII_POSCTRLWORD on page 446
Position	DINT	16#0	Assign output variable %QD of the position command value S-0-0282
Velocity	DINT	16#0	Assign output variable %QD of the positioning velocity S-0-0259

Tab. 12-30: Elements of the MB_SIII_OUT_PROC_POSITIONING structure

12.2.17 MB_SIII_IN_PROC_POS_INTERPOLATION

Brief description The data structure MB_SIII_IN_PROC_POS_INTERPOLATION contains the process input data of the function block MB_SIIIPlcDrivePosInterpolationType01 on page 440.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135 to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
Position	DINT	16#0	Assign input variable %IW of the current position S-0-0051.

Tab. 12-31: Elements of the MB_SIII_IN_PROC_POS_INTERPOLATION structure

12.2.18 MB_SIII_OUT_PROC_POS_INTERPOLATION

Brief description The data structure MB_SIII_OUT_PROC_POS_INTERPOLATION contains the process output data of the function block MB_SIIIPlcDrivePosInterpolationType01 on page 440.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134 to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
Position	DINT	16#0	Assign output variable %QD of the position command value S-0-0258.
Velocity	DINT	16#0	Assign output variable %QD of the positioning velocity S-0-0259.

Tab. 12-32: Elements of the MB_SIII_OUT_PROC_POS_INTERPOLATION structure

12.3 PowerSupply

The PowerSupply function blocks control the power supply. Sercos III is used for the communication between the control and the power supply. These function blocks can detect the internal power supply status. Additionally, they are responsible for smooth transitions between the state machines of the power supply, they set the power supply to its required state, control the operation mode, delete the power supply fault and retrieve the diagnostic information of the power supply.

The following rules apply when using function blocks for the power supply:

- If the MLC controls an inverter containing a drive and the required power supply and if this inverter is connected to the MLC control (i.e. an inverter axis is located below an MLC control node), the function block MB_SIIIPowerSupplyDriveIntegrated is used.
- If one of the extended supply unit functions **DC-DC converter** or **Microgrid mode** is used, the function block MB_SIIIPowerSupplyDcDcIslandGrid is used
- In all other cases, use the function block MB_SIIIPowerSupplyStandalone

If the power supply KMV03 is for example connected to an MLC, use the function block MB_SIIIPowerSupplyStandalone. MB_SIIIPowerSupplyStandalone is normally the most frequently used function block.

The following figure shows the state machine controlled by the function blocks of the power supply.

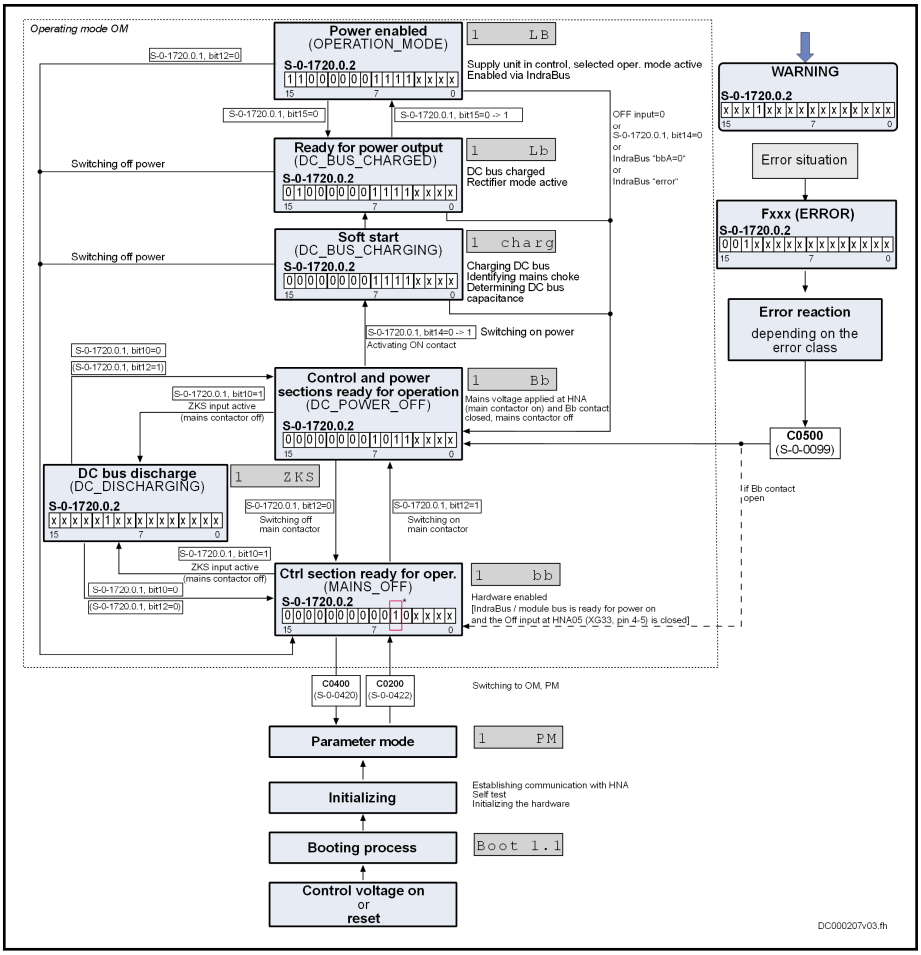


Fig. 12-17: Device control (general state machine)

For more information, refer to the documentation of the IndraDrive power supply for a general description of the power supply functionality (IndraDrive Power Supply Basic PSB-21 Functions, DOK-INDRV*-PSB-21VRS**-APRS-EN-P).

12.3.1 MB_SIIIPowerSupplyDriveIntegrated

Brief description

The MB_SIIIPowerSupplyDriveIntegrated function block monitors and controls a supply unit.

Data is exchanged between the function block and the supply unit using the Sercos field bus.

The function block is able to identify the internal supply unit state, trigger the transitions between the supply unit state machines, bring the supply unit into the required state and control the operation mode.

The function block continuously monitors the status word "InProcStatus" of the supply unit and updates the control word "OutProcCtrl" of the supply unit based on the inputs assignment.

☞ In 15V06, the function block "MB_SIIIPowerSupplyType03" was renamed in "MB_SIIIPowerSupplyDriveIntegrated". This function is unchanged.

Library:	RMB_SercosIII_Util
Group:	"PowerSupply"
Target system:	MTX 15VRS MLC 15VRS, PLC function package MLD with MPx20

Tab. 12-33: Reference table of the MB_SIIIPowerSupplyDriveIntegrated function block

Interface description

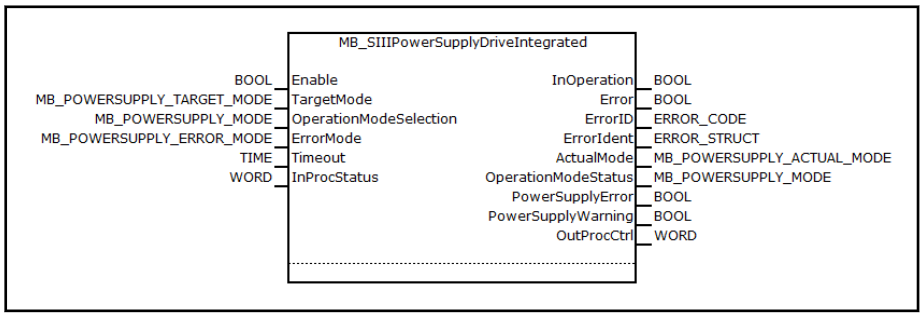


Fig. 12-18: Interface of the MB_SIIIPowerSupplyDriveIntegrated function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing enabled (cyclic, state-controlled)
	TargetMode	MB_POWERSUP-PLY_TARGET_MODE	Target mode to be reached by the supply unit MB_POWERSUPPLY_TARGET_MODE, page 474
	OperationMode-Selection	MB_POWERSUP-PLY_MODE	Operation mode of power supply MB_POWERSUPPLY_MODE, page 472
	ErrorMode	MB_POWERSUP-PLY_ERROR_MODE	Error at transition handling input for the supply unit MB_POWERSUPPLY_ERROR_MODE, page 473
	Timeout	TIME	Max. permitted time of each "ActualMode" -> "Target-Mode"
	InProcStatus	WORD	Status word of the supply unit (S-0-1720.0.2)
VAR_OUTPUT	InOperation	BOOL	The function block operates correctly. Output data is valid
	Error	BOOL	Indicates that an error occurred (see ErrorID and ErrorI-dent)
	ErrorID	ERROR_ CODE	Error cause
	ErrorIdent	ERROR_ STRUCT	Detailed error cause
	ActualMode	MB_POWERSUP-PLY_ACTUAL_MODE	Initial state of the state machine of the supply unit MB_POWERSUPPLY_ACTUAL_MODE, page 472
	OperationMo-deStatus	MB_POWERSUP-PLY_MODE	Actual operation mode of the supply unit MB_POWERSUPPLY_MODE, page 472
	PowerSupplyEr-ror	BOOL	Internal supply unit error

	Name	Type	Comment
	PowerSupply-Warning	BOOL	Internal supply unit warning
	OutProcCtrl	WORD	Supply unit control word (S-0-1720.0.1)

Tab. 12-34: Interface description of the MB_SIIIPowerSupplyDriveIntegrated function block

Min./max. and default values and input application

Name	Type	Min. value	Max. value	Default value	Application
Enable	BOOL			FALSE	Continuous
TargetMode	MB_POWERSUP-PLY_TARGET_MODE			MAINS_OFF_NO_DIS-CHARGE	Continuous
OperationMode-Selection	MB_POWERSUP-PLY_MODE			OPERATION-MODE1	Continuous
ErrorMode	MB_POWERSUP-PLY_ERROR_MODE			ERROR_TRANSI-TION_NO_DIS-CHARGE	Continuous
Timeout	TIME	N. def.	N. def.	T#10s	Continuous
InProcStatus	WORD	N. def.	N. def.	0	Continuous

Tab. 12-35: Min./max. and default values of the MB_SIIIPowerSupplyDriveIntegrated function block:

Functional Description

The function block monitors and controls the supply unit. The function block can implement different power supply functions specified in the supply unit control word.

The main task of the function block is to execute a state transition of the supply unit to the target mode.

The cyclic data exchange between the function block and the supply unit is described in the following.

- "InProcStatus" – Updates to S-0-1720.0.2 (supply unit status word)
- "OutProcCtrl" – Updates to S-0-1720.0.1 (supply unit control word)

A general description of the internal state machine of the supply unit can be found in the documentation of the IndraDrive supply unit functions.

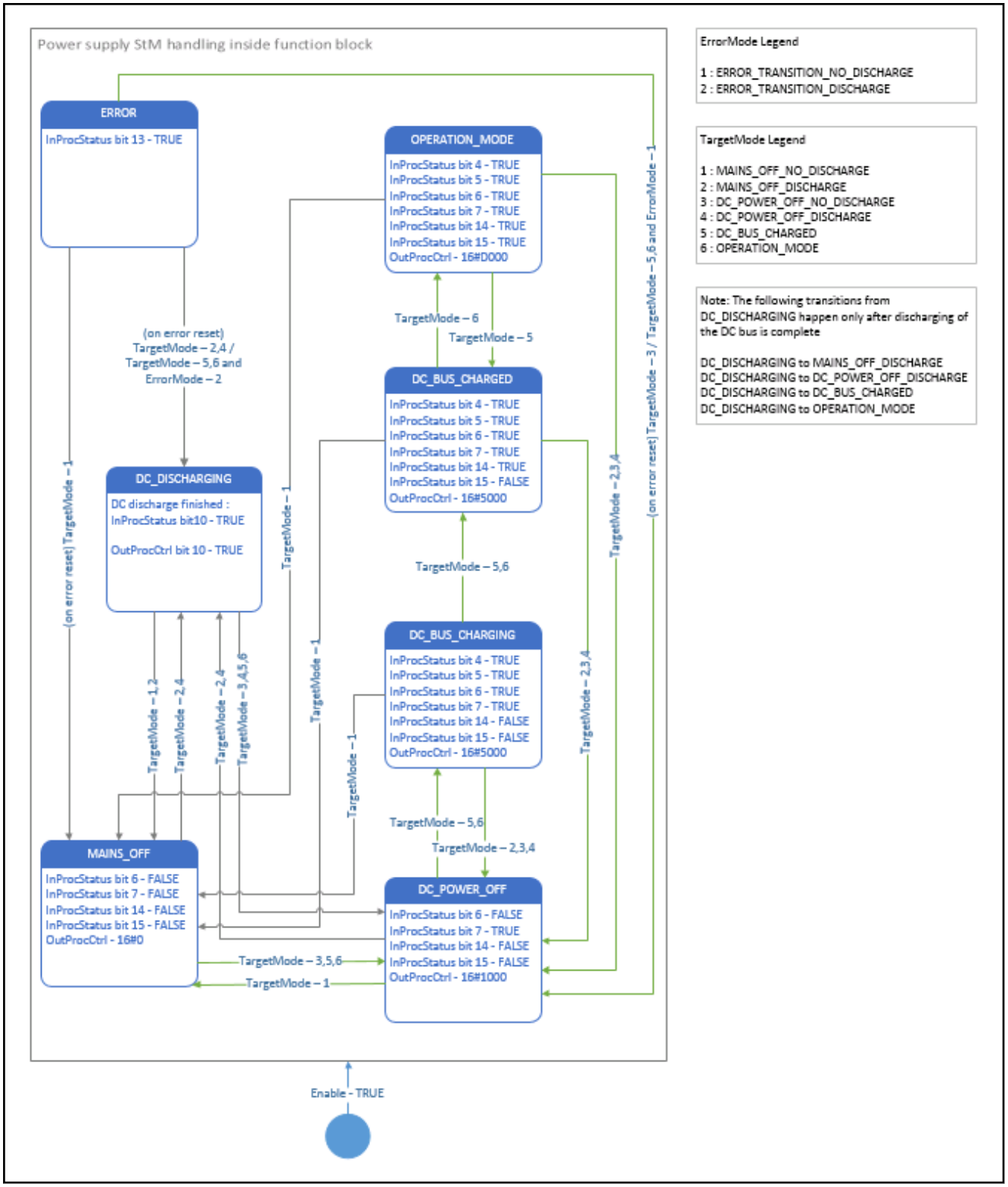


Fig. 12-19: State graph of the supply unit

The function block uses this status word to identify the current state of the state machine of the supply unit.

The target mode to be reached by the state machine of the supply unit is located at the "TargetMode" input. The function block executes the transition to "TargetMode" by controlling the internal state machine of the supply unit.

In case of a function block error, the state handling is stopped and the error is cleared.

If the supply unit goes into error state, the target and error mode decide on the next state of the supply unit.

☞ Contradictory input combinations between "ErrorMode" and "TargetMode" can occur.

In these cases, "TargetMode" dominates, irrespective of the value of the "ErrorMode" input. See the following example.

- ActualMode = _ERROR
- TargetMode = DC_POWER_OFF_NO_DISCHARGE
- ErrorMode = ERROR_TRANSITION_DISCHARGE

If the power supply leaves the "Error" state, "TargetMode" requires for the power supply to go to state "DC Power OFF" without discharging the DC bus.

"ErrorMode" requires the power supply to discharge the DC bus before switching to the next state.

In this case, "TargetMode" dominates and the power supply is switched to mode "DC Power Off" without discharging the DC bus.

This has to be taken into consideration by the programmer when programming the application.

The time, the supply unit needs to reach the target mode (from the current mode), is monitored.

If the time expires, the "Timeout" input reports an error and the state machine handling is stopped.

If the "Enable" input of the function block is reset, the function block detects the current supply unit state and decides whether it is requires to switch off the supply unit (DC).

The "OperationModeSelection" input is immediately and continuously displayed on bits 8 and 9 of the control word.

The function block outputs are refreshed upon each PLC scan by the "InProcStatus" input. Each status word change is immediately output at the outputs.

The outputs "Error", "ErrorID" and "ErrorIdent" contain the information on the error.

Troubleshooting

The function block can go to an error state due to invalid inputs or an expired timeout timer. In both cases, the function block reports an error and cancels the operation.

The user can fixed the error cause, reset the "Enable" input and set the input again (standard procedure when resetting errors).

The function block subsequently continues. Refer to the following tables for available error codes.

In addition to the errors of the internally used function blocks IL_SIIICommand and IL_SIIIRemoteSlaveState, the following function block errors are listed in the error tableF_RELATED_TABLE (ERROR_TABLE = 16#0170).

STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_INVALID_ERROR	16#00002990	TargetMode	TargetMode input is invalid
INPUT_INVALID_ERROR	16#00002991	OperationMo-deSelection	OperationModeSelection input invalid
INPUT_INVALID_ERROR	16#00002992	ErrorMode	ErrorMode input is invalid
STATE_MACHINE_ERROR	16#00002993	0	Timeout (actual mode -> target mode)

Tab. 12-36: Error codes of the MB_SIIIPowerSupplyDriveIntegrated function block

Use case with MB_SIIIPowerSup-plyDriveIntegrated

Two IndraDrive XMS are connected to one XCS IndraDrive with integrated supply unit.

The function block MB_SIIIPowerSupplyDriveIntegrated controls the supply unit XCS. The inverters XMS are enabled using the outputs of this function block (switched from "bb" to "Ab").

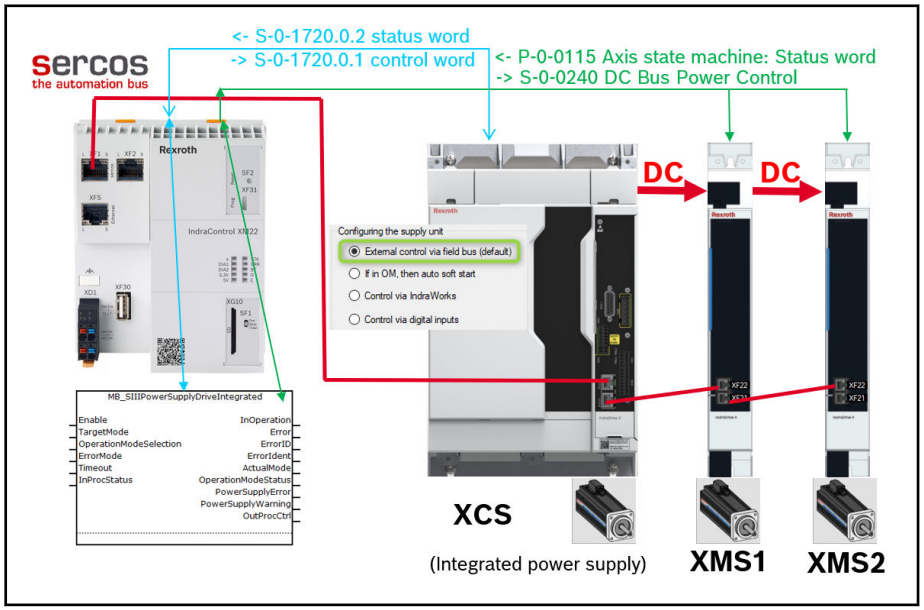


Fig. 12-20: Schematic diagram: Integrated supply unit XCS supplies two XMS drives

Prerequisite: The supply unit in the XCS is configured to "External activation via field bus".

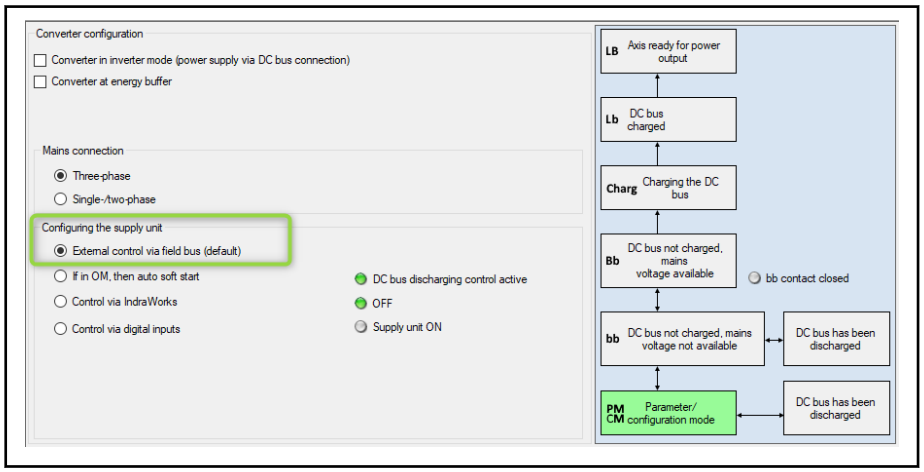


Fig. 12-21: Configuration of the integrated supply unit

The following parameters have to be configured cyclically for the XCS:

- MDT: S-0-1720.0.1, "Supply unit control word"
- AT: S-0-1720.0.2, "Power supply status word"

For XMS drives, it is sufficient to configure the following data in the signal control word (MDT) or signal status word (AT):

- MDT (AxisData[].dwUserCmdDataA_q): Bit 0 "DC bus power control" of the parameter S-0-0240, "Intermediate circuit supply: Control word"
- AT (AxisData[].dwUserActualDataA_i): Bit 0 "Ready for Power Supply: Enabling power supply of inverter" of the parameter P-0-0115, "Axis state machine: Status word, axis"

The following PLC program example assumes that the data containers "A" are used in both directions.

All other containers "from B to F" can be used instead. Modify the PLC program respectively.



Use the function block "MB_GetCyclicParameterHandle" and the functions "MB_ReadCyclicParameter" and "MB_WriteCyclicParameter" (refer to the "ML_TechMotionBase" library) to implement the PLC program irrespective of the data containers "A to F" used.

If the XMS or XCS drives are implemented as PLC devices, configure the same parameters cyclically and map the variables to these parameters. The instructions are commented in the following example and labeled with "drive as PLC device".

Variable definition

```
fbMB_SIIIPowerSupply: MB_SIIIPowerSupplyDriveIntegrated;
bEnablePower: BOOL:=FALSE;
```

Initialization start (example GATcompact, Machine.actInit_entry)

```
// switch off power supply
bEnablePower := FALSE;
```

Initialization end (example GATcompact, Machine.actInit_active)

```
IF (ImcStatus.Admin._OpModeAck = IMC_MODE_STATUS_BB) AND (TE_AxisInterfaceType01Prog.InitDone = TRUE) THEN
   // ********************************************
   // Now we reached BB (operating mode)
   // Add your code for writing the parameters (in BB) here
   // ********************************************
   bEnablePower := TRUE;
   IF fbMB_SIIIPowerSupply.ActualMode = MB_POWERSUPPLY_ACTUAL_MODE.OPERATION_MODE THEN
     uiStateInit := uiSTATE_INIT_PREPARE_COLDSTART;
     bInitDone   := TRUE;   // Set bInitDone if the initialization is done
   END_IF
END_IF
```

Called cyclically (example GATcompact, Machine.actCyclic_active)

```
//-----------------------------------------------------------------------------------------------
//------------------------------ handle power-supply  -------------------------------------
//-----------------------------------------------------------------------------------------------
// copy status word from bus to FB
fbMB_SIIIPowerSupply.InProcStatus := AxisData[XCS.AxisNo].dwUserActualDataA_i.UINT_; // motion axis
// fbMB_SIIIPowerSupply.InProcStatus := PlcDrive_InProcStatus; // drive as PLC device
// Check that all devices are ready for operation ("bb"-relais is closed)
  IF (AxisData[XMS1.AxisNo].wUserActualDataBitA_i.0 AND AxisData[XMS2.AxisNo].wUserActualDataBitA_i.0)
          AND bEnablePower THEN
    // switching direct to OPERATION_MODE when drives are ready
    fbMB_PowerSupply.TargetMode := MB_POWERSUPPLY_TARGET_MODE.OPERATION_MODE;
```

```
  ELSE
    // switching OFF when drives are not ready or bEnablePower is FALSE
    fbMB_PowerSupply.TargetMode := MB_POWERSUPPLY_TARGET_MODE.MAINS_OFF_NO_DISCHARGE;
  END_IF

fbMB_SIIIPowerSupply(
  Enable:= bEnablePower,
  TargetMode:= ,
  OperationModeSelection:= ,
  ErrorMode:= ,
  Timeout:= ,
  OperationModeStatus=> ,
  PowerSupplyError=> ,
  PowerSupplyWarning=> );
IF fbMB_SIIIPowerSupply.Error THEN
  bEnablePower := FALSE;
  mSetFBError(fbMB_SIIIPowerSupply.ErrorID, fbMB_SIIIPowerSupply.ErrorIdent);
END_IF;
// copy control word from FB to bus
AxisData[XCS.AxisNo].dwUserCmdDataA_q.UINT_ := fbMB_SIIIPowerSupply.OutProcCtrl; // motion axis
// PlcDrive_OutProcCtrl := fbMB_SIIIPowerSupply.OutProcCtrl; // drive as PLC device
// copy output of FB to DC Bus Power Control, bit 0 of all XMS
IF fbMB_SIIIPowerSupply.ActualMode = MB_POWERSUPPLY_ACTUAL_MODE.OPERATION_MODE THEN
  // activate power in power supply control word
  AxisData[XMS1.AxisNo].wUserCmdDataBitA_q.0 := TRUE; // motion axis 1
  AxisData[XMS2.AxisNo].wUserCmdDataBitA_q.0 := TRUE; // motion axis 2
  // aruiPowerSupplyControlWord[1].0 :=TRUE; // drive as PLC device
ELSE
  // switch off power in DC Bus Power Control
  AxisData[XMS1.AxisNo].wUserCmdDataBitA_q.0 := FALSE;
  AxisData[XMS2.AxisNo].wUserCmdDataBitA_q.0 := FALSE;
  // aruiPowerSupplyControlWord[1].0 :=FALSE; // drive as PLC device
END_IF
```

For facilitation purposes, it is directly switched to "OPERATION_MODE" at the end of the initialization mode. For interim states, see fig. 12-19 "State graph of the supply unit" on page 453. This logic corresponds to the automatic start of the supply unit.

Use case: Automatic without MB_SIIIPowerSupplyDriveIntegrated

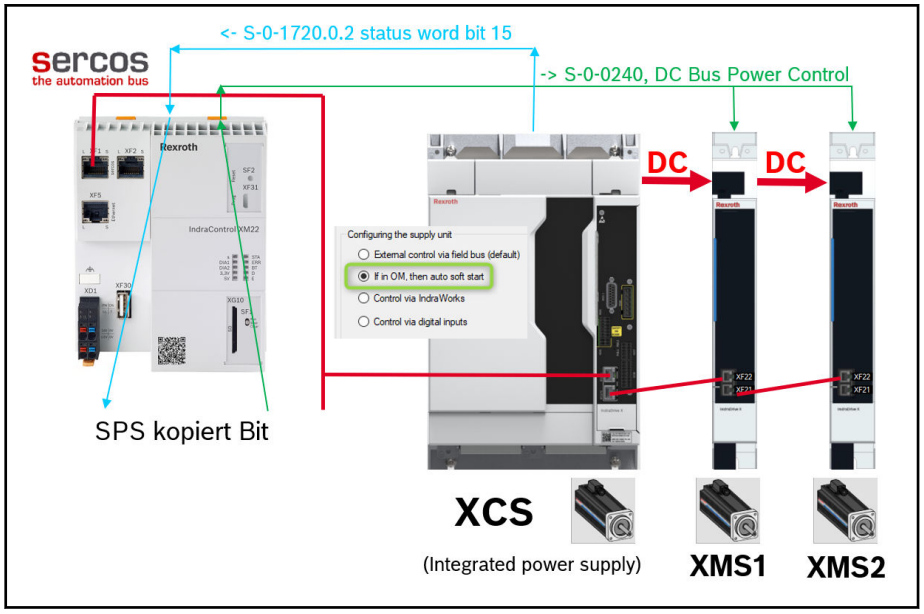


Fig. 12-22: Schematic diagram: Integrated supply unit XCS in automatic mode supplies two XMS drives

As an alternative to enable the supply unit using the function block MB_SIIIPowerSupplyDriveIntegrated, the automatic start of the supply unit can be used.

Select the option "if in OM, then auto start" in the dialog fig. 12-21 "Configuration of the integrated supply unit" on page 455.

For the XCS, it is sufficient that bit 15 "Enabling" of the parameter S-0-1720.0.2, "Power supply status word" is to implement into the signal status word.

In the PLC, bit 15 of S-0-1720.0.2, "Supply unit status word" of XCS is copied to bit 0 "DC bus power control" of the parameter S-0-0240, "DC bus supply: Control word" of all XMS drives.

Called cyclically (example GATcompact, Machine.actCyclic_active)

```
// Activate power for axis XMS, bit 0 of controlword S-0-0240 and bit 15 of statusword 1720.0.2 is configured
AxisData[XMS1.AxisNo].wUserCmdDataBitA_q.0 := AxisData[XCS.AxisNo].wUserActualDataBitA_i.0;
AxisData[XMS2.AxisNo].wUserCmdDataBitA_q.0 := AxisData[XCS.AxisNo].wUserActualDataBitA_i.0;
```

12.3.2 MB_SIIIPowerSupplyStandalone

Brief description

The MB_SIIIPowerSupplyStandalone function block monitors and controls a supply unit.

Data is exchanged between the function block and the supply unit using the Sercos field bus.

The function block is able to identify the internal supply unit state, trigger the transitions between the supply unit state machines, bring the supply unit into the required state and control the operation mode.

The function block continuously monitors the status word "InProcStatus" of the supply unit and updates the control word "OutProcCtrl" of the supply unit based on the inputs assignment.

☞ In 15V06, the function block "MB_SIIIPowerSupplyType04" was renamed in "MB_SIIIPowerSupplyStandalone". This function is unchanged.

Library:	RMB_SercosIII_Util
Group:	"PowerSupply"
Target system:	MTX 15VRS MLC 15VRS, PLC function package MLD with MPx20

Tab. 12-37: Reference table of the MB_SIIIPowerSupplyStandalone function block

Interface description

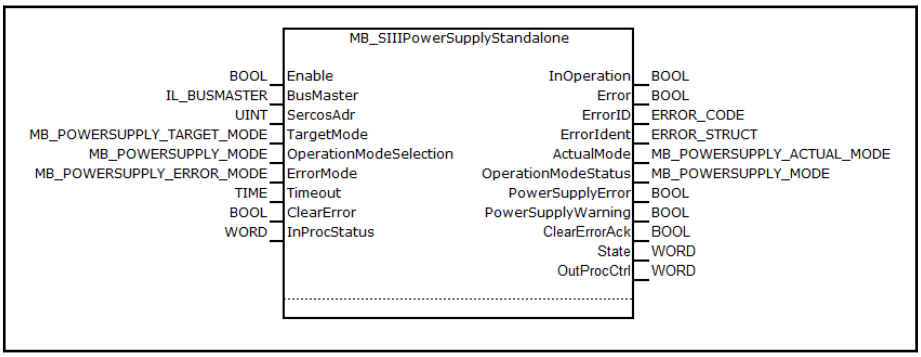


Fig. 12-23: Interface of the MB_SIIIPowerSupplyStandalone function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing enabled (cyclic, state-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, the bus master ID is IL_BUSMASTER_0.
	SercosAdr	UINT	Sercos III address of the slave node
	TargetMode	MB_POWERSUPPLY_TARGET_MODE	Target mode to be reached by the supply unit MB_POWERSUPPLY_TARGET_MODE, page 474
	OperationModeSelection	MB_POWERSUPPLY_MODE	Operation mode of power supply MB_POWERSUPPLY_MODE, page 472
	ErrorMode	MB_POWERSUPPLY_ERROR_MODE	Error at transition handling input for the supply unit MB_POWERSUPPLY_ERROR_MODE, page 473
	ClearError	BOOL	Clears an error in the drive by the "Clear error" command (S-0-0099). After the error has been cleared, the selected enabling can be activated again.
	Timeout	TIME	Max. permitted time of each "ActualMode" -> "TargetMode"
	InProcStatus	WORD	Status word of the supply unit (S-0-1720.0.2)
VAR_OUTPUT	InOperation	BOOL	The function block operates correctly. Output data is valid
	Error	BOOL	Indicates that an error occurred (see ErrorID and ErrorIdent)
	ErrorID	ERROR_ CODE	Error cause
	ErrorIdent	ERROR_ STRUCT	Detailed error cause
	ActualMode	MB_POWERSUPPLY_ACTUAL_MODE	Initial state of the state machine of the supply unit MB_POWERSUPPLY_ACTUAL_MODE, page 472
	OperationModeStatus	MB_POWERSUPPLY_MODE	Actual operation mode of the supply unit MB_POWERSUPPLY_MODE, page 472
	PowerSupplyError	BOOL	Internal supply unit error
	PowerSupplyWarning	BOOL	Internal supply unit warning
	ClearErrorAck	BOOL	Confirmation that the command "Clear error" S-0-0099 was executed successfully.
	State	WORD	Device status of IL_SIIIRemoteSlaveState
	OutProcCtrl	WORD	Supply unit control word (S-0-1720.0.1)

Tab. 12-38: Interface description of the MB_SIIIPowerSupplyStandalone function block

Min./max. and default values and input application

Name	Type	Min. value	Max. value	Default value	Application
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	N. def.	N. def.	IL_BUSMAS-TER_0	
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	System-specific	0	
TargetMode	MB_POWERSUP-PLY_TARGET_MODE			MAINS_OFF_NO_DIS-CHARGE	Continuous
OperationMode-Selection	MB_POWERSUP-PLY_MODE			OPERATION-MODE1	Continuous
ErrorMode	MB_POWERSUP-PLY_ERROR_MODE			ERROR_TRANSI-TION_NO_DIS-CHARGE	Continuous
ClearError	BOOL			FALSE	Continuous
Timeout	TIME	N. def.	N. def.	T#10s	Continuous
InProcStatus	WORD	N. def.	N. def.	0	Continuous

Tab. 12-39: Min./max. and default values of the MB_SIIIPowerSupplyStandalone function block:

Functional Description

The function block monitors and controls the supply unit. The function block can implement different supply unit functions specified in the supply unit control word.

The main task of the function block is to execute a state transition of the supply unit to the target mode.

The cyclic data exchange between the function block and the supply unit is described in the following.

- "InProcStatus" – Updates to S-0-1720.0.2 (supply unit status word)
- "OutProcCtrl" – Updates to S-0-1720.0.1 (supply unit control word)

A general description of the internal state machine of the supply unit can be found in the documentation of the IndraDrive supply unit functions.

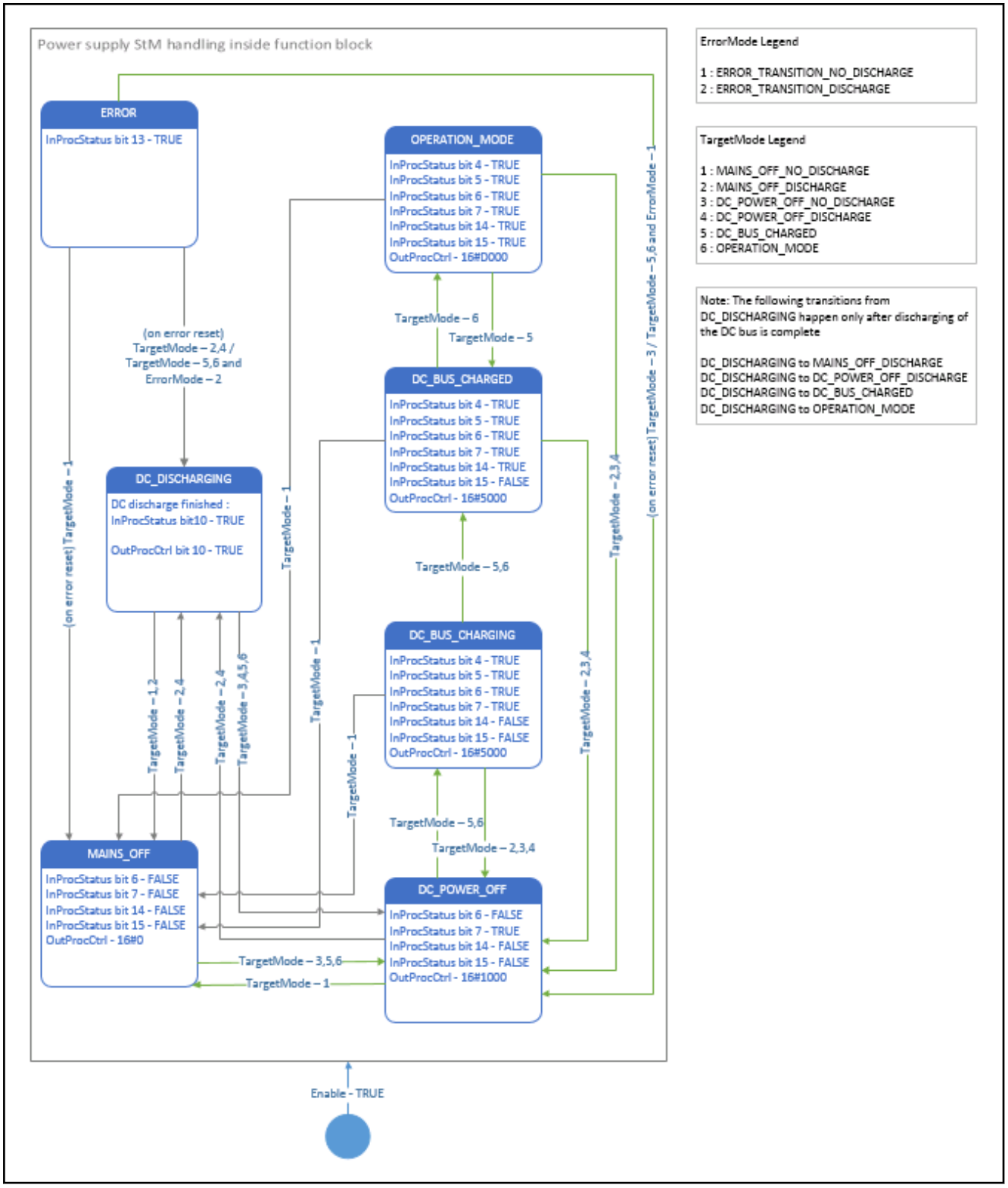


Fig. 12-24: State graph of the supply unit

The function block uses this status word to identify the current state of the state machine of the power supplies.

The target mode to be reached by the state machine of the supply unit is already available at the "TargetMode" input.

The traveled path to reach the target mode is known to the function block. Thus, the function block triggers the supply unit transition to the next immediate state. This transition to the next immediate state continues until the supply unit has reached the target mode after different states.

In case of a function block error, the state handling is stopped and the error is cleared.

If the supply unit goes into error state, the target and error mode decide on the next state of the supply unit.

☞ Contradictory input combinations between "ErrorMode" and "TargetMode" can occur.

In these cases, "TargetMode" dominates, irrespective of the value of the "ErrorMode" input. See the following example.

- ActualMode = _ERROR
- TargetMode = DC_POWER_OFF_NO_DISCHARGE
- ErrorMode = ERROR_TRANSITION_DISCHARGE

If the power supply leaves the "Error" state, "TargetMode" requires for the power supply to go to state "DC Power OFF" without discharging the DC bus.

"ErrorMode" requires the power supply to discharge the DC bus before switching to the next state.

In this case, "TargetMode" dominates and the power supply is switched to mode "DC Power Off" without discharging the DC bus.

This has to be taken into consideration by the programmer when programming the application.

The time, the supply unit needs to reach the target mode (from the current mode), is monitored.

If the time expires, the "Timeout" input reports an error and the state machine handling is stopped.

If the "Enable" input of the function block is reset, the function block detects the current supply unit state and decides whether it is requires to switch off the supply unit (DC).

In this case, the control word is set to set the supply unit to DC Power Off mode (if required).

The outputs "Error", "ErrorID" and "ErrorIdent" contain the information on the error.

The operation mode selection input is immediately and continuously displayed on bits 8 and 9 of the control word. The function block outputs are refreshed upon each PLC scan by the "InProcStatus" input. Each status word change is immediately output at the outputs.

Furthermore, the supply unit error can be cleared via the "ClearError" input. The rising edge of this inputs triggers the "Clear error" command S-0-0099 for the supply unit. The status of a successfully cleared error is output via the "ClearErrorAck" output of the function block.

The diagnostic information on the supply unit is continuously updated at the "State" function block output. The IL_SIIIRemoteSlaveState function block performs the update.

Troubleshooting

The function block can go to an error state due to invalid inputs or an expired timeout timer. In both cases, the function block generates an error and cancels the operation.

The user can fixed the error cause, reset the "Enable" input and set the input again (standard procedure when resetting errors).

The function block subsequently continues. Refer to the following tables for available error codes.

In addition to the errors of the internally used function blocks IL_SIIICommand and IL_SIIIRemoteSlaveState, the following function block errors are listed in the error tableF_RELATED_TABLE (ERROR_TABLE = 16#0170).

STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_INVALID_ERROR	16#00002990	TargetMode	TargetMode input is invalid
INPUT_INVALID_ERROR	16#00002991	OperationModeSelection	OperationModeSelection input invalid
INPUT_INVALID_ERROR	16#00002992	ErrorMode	ErrorMode input is invalid
STATE_MACHINE_ERROR	16#00002993	0	Timeout (actual mode -> target mode)

Tab. 12-40: Error codes of the MB_SIIIPowerSupplyStandalone function block

Use case with MB_SIIIPowerSupplyStandalone

Two IndraDrive XMS are connected to one XVR supply unit.

The function block MB_SIIIPowerSupplyStandalone controls the supply unit XVR. The drives of the inverters XMS are enabled using the outputs of this function block (switched from "bb" to "Ab").

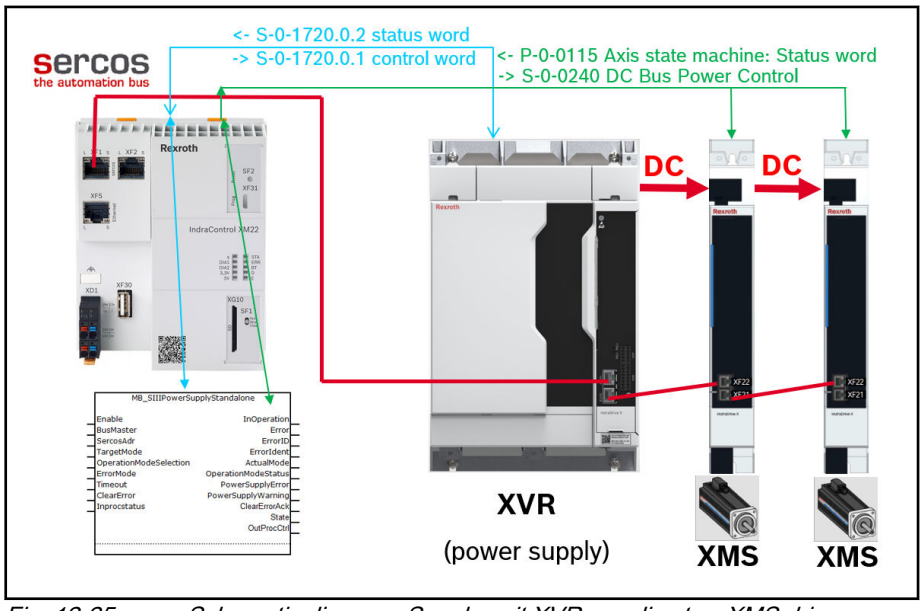


Fig. 12-25: Schematic diagram: Supply unit XVR supplies two XMS drives

The following PLC program example can also be used for the power supplies HMV and KMV. Copying the supply unit status word to the bit "DC bus power control" of the slaves is omitted. In this case, the information is sent via the module bus.

The following parameters have to be configured cyclically at the supply unit XVR and mapped to the PLC variables:

- MDT: S-0-1720.0.1, "Supply unit control word"

- AT: S-0-1720.0.2, "Power supply status word"

For XMS drives, it is sufficient to configure the following data in the signal control word (MDT) or signal status word (AT):

- MDT (AxisData[].dwUserCmdDataA_q): Bit 0 "DC bus power control" of the parameter S-0-0240, "Intermediate circuit supply: Control word"
- AT (AxisData[].dwUserActualDataA_i): Bit 0 "Ready for Power Supply: Enabling power supply of inverter" of the parameter P-0-0115, "Axis state machine: Status word, axis"

The following PLC program example assumes that the data containers "A" are used in both directions.

All other containers "from B to D" can be used instead. Modify the PLC program respectively.



Use the function block "MB_GetCyclicParameterHandle" and the functions "MB_ReadCyclicParameter" and "MB_WriteCyclicParameter" (refer to the "ML_TechMotionBase" library) to implement the PLC program irrespective of the data containers "A to D" used.

If the XMS drives are implemented as PLC devices, configure the same parameters cyclically and map the variables to these parameters.

The instructions are commented in the following example and labeled with "drive as PLC device".

Variable definition

```
fbMB_SIIIPowerSupply: MB_SIIIPowerSupplyStandalone;
bEnablePower: BOOL:=FALSE;
```

Initialization start (example GATcompact, Machine.actInit_entry)

```
// switch off power supply
bEnablePower := FALSE;
```

Initialization end (example GATcompact, Machine.actInit_active)

```
IF (ImcStatus.Admin._OpModeAck = IMC_MODE_STATUS_BB) AND (TE_AxisInterfaceType01Prog.InitDone = TRUE) THEN
   // ************************************************
   // Now we reached BB (operating mode)
   // Add your code for writing the parameters (in BB) here
   // ************************************************
   bEnablePower := TRUE;
   IF fbMB_SIIIPowerSupply.ActualMode = MB_POWERSUPPLY_ACTUAL_MODE.OPERATION_MODE THEN
     uiStateInit := uiSTATE_INIT_PREPARE_COLDSTART;
     bInitDone   := TRUE;   // Set bInitDone if the initialization is done
   END_IF
END_IF
```

Called cyclically (example GATcompact, Machine.actCyclic_active)

```
//-------------------------------------------------------------------------------------------------------
//------------------------------ handle power-supply  -----------------------------------
//-------------------------------------------------------------------------------------------------------
// copy status word from bus to FB
fbMB_SIIIPowerSupply.InProcStatus := XVR_InProcStatus;// drive as PLC device
IF (AxisData[XMS1.AxisNo].wUserActualDataBitA_i.0 AND AxisData[XMS2.AxisNo].wUserActualDataBitA_i.0)
          AND bEnablePower THEN
    // switching direct to OPERATION_MODE when drives are ready
    fbMB_PowerSupply.TargetMode := MB_POWERSUPPLY_TARGET_MODE.OPERATION_MODE;
  ELSE
    // switching OFF when drives are not ready or bEnablePower is FALSE
    fbMB_PowerSupply.TargetMode := MB_POWERSUPPLY_TARGET_MODE.MAINS_OFF_NO_DISCHARGE;
  END_IF
fbMB_SIIIPowerSupply(
```

```
  Enable:= bEnablePower,
  BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0,
  SercosAdr:= 42,
  TargetMode:= ,
  OperationModeSelection:= ,
  ErrorMode:= ,
  Timeout:= ,
  ClearError:= PlcProg.fbMachine.ClearError,
  OperationModeStatus=> ,
  PowerSupplyError=> ,
  PowerSupplyWarning=> ,
  ClearErrorAck=> ,
  State=> );
IF fbMB_SIIIPowerSupply.Error THEN
  bEnablePower := FALSE;
  mSetFBError(fbMB_SIIIPowerSupply.ErrorID, fbMB_SIIIPowerSupply.ErrorIdent);
END_IF
// copy control word from FB to bus
XVR_OutProcCtrl := fbMB_SIIIPowerSupply.OutProcCtrl; // drive as PLC device
// copy output of FB to DC Bus Power Control, bit 0 of all XMS
IF fbMB_SIIIPowerSupply.ActualMode = MB_POWERSUPPLY_ACTUAL_MODE.OPERATION_MODE THEN
  // activate power in power supply control word
  AxisData[XMS1.AxisNo].wUserCmdDataBitA_q.0 := TRUE; // motion axis 1
  AxisData[XMS2.AxisNo].wUserCmdDataBitA_q.0 := TRUE; // motion axis 2
  // aruiPowerSupplyControlWord[1].15 :=TRUE; // drive as PLC device
ELSE
  // switch off power in power supply control word
  AxisData[XMS1.AxisNo].wUserCmdDataBitA_q.0 := FALSE;
  AxisData[XMS2.AxisNo].wUserCmdDataBitA_q.0 := FALSE;
  // aruiPowerSupplyControlWord[1].15 :=FALSE; // drive as PLC device
END_IF
```

For facilitation purposes, it is directly switched to "OPERATION_MODE" at the end of the initialization mode. For interim states, see fig. 12-24 "State graph of the supply unit" on page 461.

12.3.3 MB_SIIIPowerSupplyDcDcIslandGrid

Brief description

The MB_SIIIPowerSupplyDcDcIslandGrid function block monitors and controls a supply unit with one of the following function packages:

- **DCE** ->"DC-DC converter" for the DC-DC converter mode
- **MSE** ->"Grid function" for microgrid mode

The cyclic data exchange between the function block and the supply unit is described in the following.

The function block is able to identify the internal supply unit state, trigger the transitions between the supply unit state machines, bring the supply unit into the required state and control the operation mode.

The function block continuously monitors the status word "InProcStatus" of the supply unit and updates the control word "OutProcCtrl" of the supply unit based on the inputs assignment.

Library:	RMB_SercosIII_Util
Group:	"PowerSupply"
Target system:	MTX 15VRS MLC 15VRS, PLC function package MLD with MPx20

Tab. 12-41: Reference table of the MB_SIIIPowerSupplyDcDcIslandGrid function block

Interface description

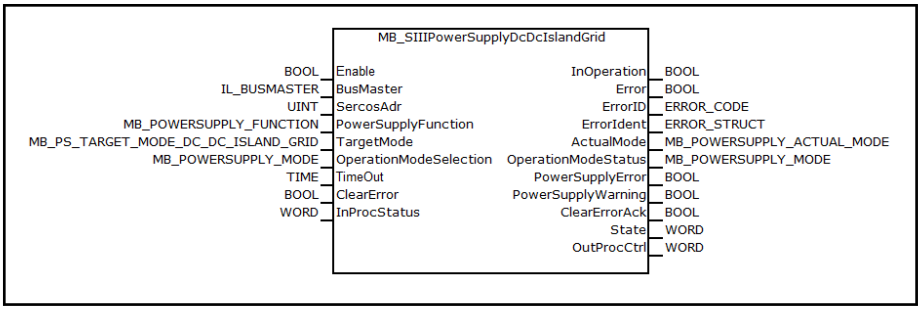


Fig. 12-26: Interface of the MB_SIIIPowerSupplyDcDcIslandGrid function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing enabled (cyclic, state-controlled)
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, the bus master ID is IL_BUSMASTER_0.
	SercosAdr	UINT	Sercos III address of the slave node
	PowerSupply-Function	MB_POWERSUPPLY_FUNCTION	Extended supply unit function and additional function packages of the supply unit are required. MB_POWERSUPPLY_FUNCTION, page 475
	TargetMode	MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID	Target mode to be reached by the supply unit MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID, page 475
	OperationMode-Selection	MB_POWERSUPPLY_MODE	Operation mode of the supply unit MB_POWERSUPPLY_MODE, page 472
	Timeout	TIME	Max. permitted time of each "ActualMode" -> "TargetMode"
	ClearError	BOOL	Clearing an error in the drive using the "Clear error" command S-0-0099. After clearing, activate the selected enabling.
	InProcStatus	WORD	Status word of the supply unit (S-0-1720.0.2)
VAR_OUTPUT	InOperation	BOOL	The function block operates correctly. The output data is valid
	Error	BOOL	Indicates that an error occurred (see ErrorID and ErrorIdent)
	ErrorID	ERROR_ CODE	Error cause
	ErrorIdent	ERROR_ STRUCT	Detailed error cause
	ActualMode	MB_POWERSUPPLY_ACTUAL_MODE	Initial state of the state machine of the supply unit MB_POWERSUPPLY_ACTUAL_MODE, page 472
	OperationMo-deStatus	MB_POWERSUPPLY_MODE	Actual operation mode of the supply unit MB_POWERSUPPLY_MODE, page 472
	PowerSupplyEr-ror	BOOL	Internal supply unit error

	Name	Type	Comment
	PowerSupply-Warning	BOOL	Internal supply unit warning
	ClearErrorAck	BOOL	Confirmation that the command "Clear error" S-0-0099 was executed successfully.
	State	WORD	Device status of IL_SIIIRemoteSlaveState
	OutProcCtrl	WORD	Supply unit control word (S-0-1720.0.1)

Tab. 12-42: Interface description of the MB_SIIIPowerSupplyDcDcIslandGrid function block

Min./max. and default values and input application

Name	Type	Min. value	Max. value	Default value	Application
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	N. def.	N. def.	IL_BUSMAS-TER_0	Rising edge at “Enable”
SercosAdr	UINT	0 (MLD: local axis) 1 (other sys-tems)	System-specific	0	Rising edge at “Enable”
PowerSupply-Function	MB_POWERSUP-PLY_FUNCTION			PS_FUNC-TION_DC_DC	Rising edge at “Enable”
TargetMode	MB_PS_TAR-GET_MODE_DC_DC_IS LAND_GRID			MAINS_OFF	Continuous
OperationMode-Selection	MB_POWERSUP-PLY_MODE			OPERATION-MODE1	Continuous
ClearError	BOOL			FALSE	Continuous
Timeout	TIME	N. def.	N. def.	T#10s	Continuous
InProcStatus	WORD	N. def.	N. def.	0	Continuous

Tab. 12-43: Min./max. and default values of the MB_SIIIPowerSupplyDcDcIslandGrid function block:

Functional description

The function block monitors and controls the supply unit. The function block can implement different supply unit functions specified in the supply unit control word.

The main task of the function block is to get the supply unit to the desired target mode.

The cyclic data exchange between the function block and the supply unit is described in the following.

- "InProcStatus" – Updates to S-0-1720.0.2 (supply unit status word)
- "OutProcCtrl" – Updates to S-0-1720.0.1 (supply unit control word)

A general description of the internal state machine of the supply unit can be found in the documentation of the IndraDrive supply unit functions.

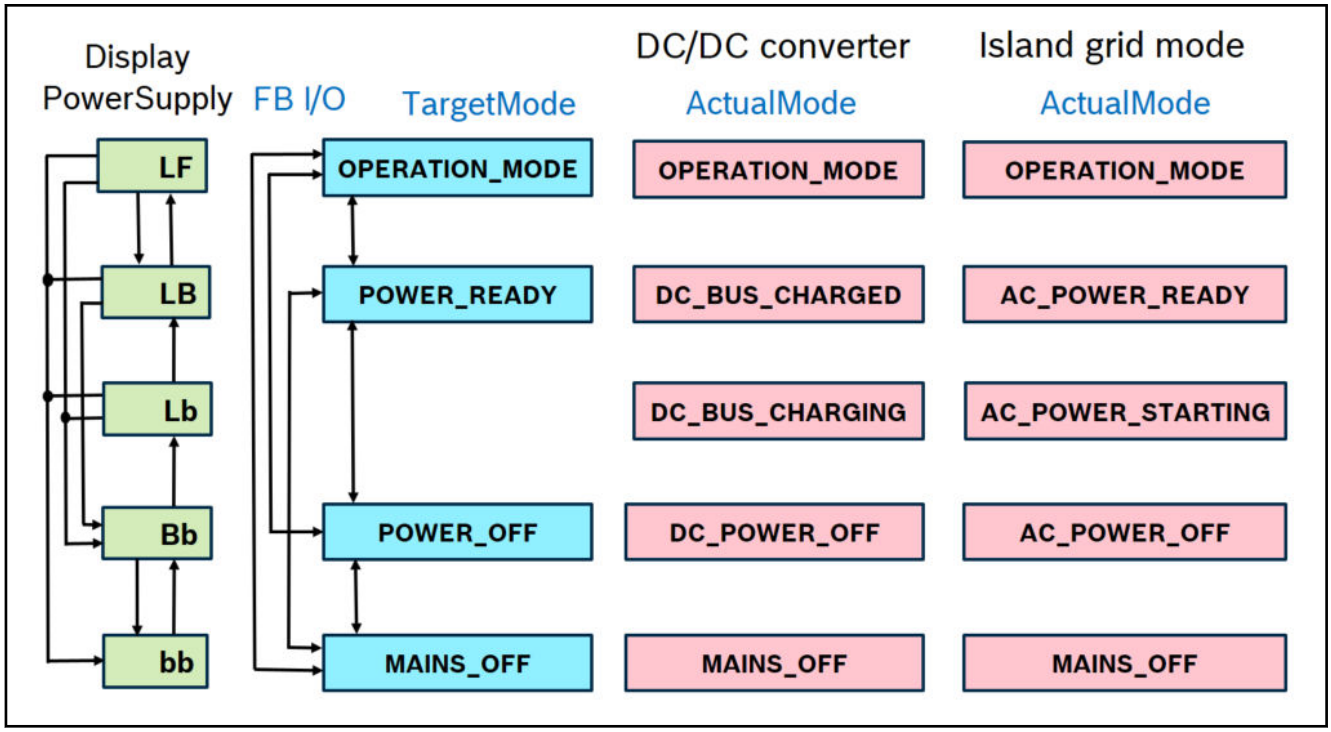


Fig. 12-27: Simplified state graph of a supply unit with the respective function block inputs/outputs

The function block uses the supply unit status word to identify the current state of the state machine of the supply unit.

The current state is output at the "ActualMode" output. It depends either on the **DC/DC converter** function or on the **microgrid mode** function selected at the "PowerSupplyFunction" input (refer to the previous state graph).

The target mode to be reached by the state machine of the supply unit is specified using the "TargetMode" input. The previous path to reach the target mode is stored in the function block. The function block performs all required state transitions up to the target mode.

In case of a function block error, the state handling is stopped and the error is signaled at the "Error" output.

The time, the supply unit needs to reach the target mode from the current mode is monitored using the "Timeout" input. If the timeout time expires, the function block reports an error and the state machine handling is stopped.

If the "Enable" input of the function block is reset, the function block switches off the supply unit. The control word is reset to set the supply unit to the Mains Off mode.

The outputs "Error", "ErrorID" and "ErrorIdent" contain the information on the error.

The operation mode selection input is continuously output on the bits 8 and 9 of the control word. The "InProcStatus" input refreshes the function block outputs upon each function block call. Each status word change is immediately output at the outputs.

Furthermore, a supply unit error can be cleared via the "ClearError" input. The rising edge of t his inputs triggers the "Clear error" command S-0-0099 for the supply unit. The status of a successfully cleared error is output via the "ClearErrorAck" output of the function block.

The diagnostic information on the supply unit is continuously updated at the "State" function block output.

Troubleshooting

The function block can go to an error state due to invalid inputs or an expired timeout timer. In both cases, the function block generates an error and cancels the operation.

The user can fixed the error cause, reset the "Enable" input and set the input again (standard procedure when resetting errors).

The function block subsequently continues. Refer to the following tables for available error codes.

In addition to the errors of the internally used function blocks IL_SIIICommand, IL_SIIIReadParameter and IL_SIIIRemoteSlaveState, the following function block errors are listed in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Comment
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_INVALID_ERROR	16#00002990	TargetMode	TargetMode input is invalid
INPUT_INVALID_ERROR	16#00002991	OperationModeSelection	OperationModeSelection input invalid
STATE_MACHINE_ERROR	16#00002993	0	Timeout (actual mode -> target mode)
INPUT_INVALID_ERROR	16#00002994	PowerSupplyFunction	The PowerSupplyFunction input is invalid

Tab. 12-44: Error codes of the MB_SIIIPowerSupplyDcDcIslandGrid function block

Use case with MB_SIIIPowerSupplyDcDcIslandGrid

Use a DC/DC converter to charge a battery.

Use the MB_SIIIPowerSupplyDcDcIslandGrid function block to control the inverter HMU05.

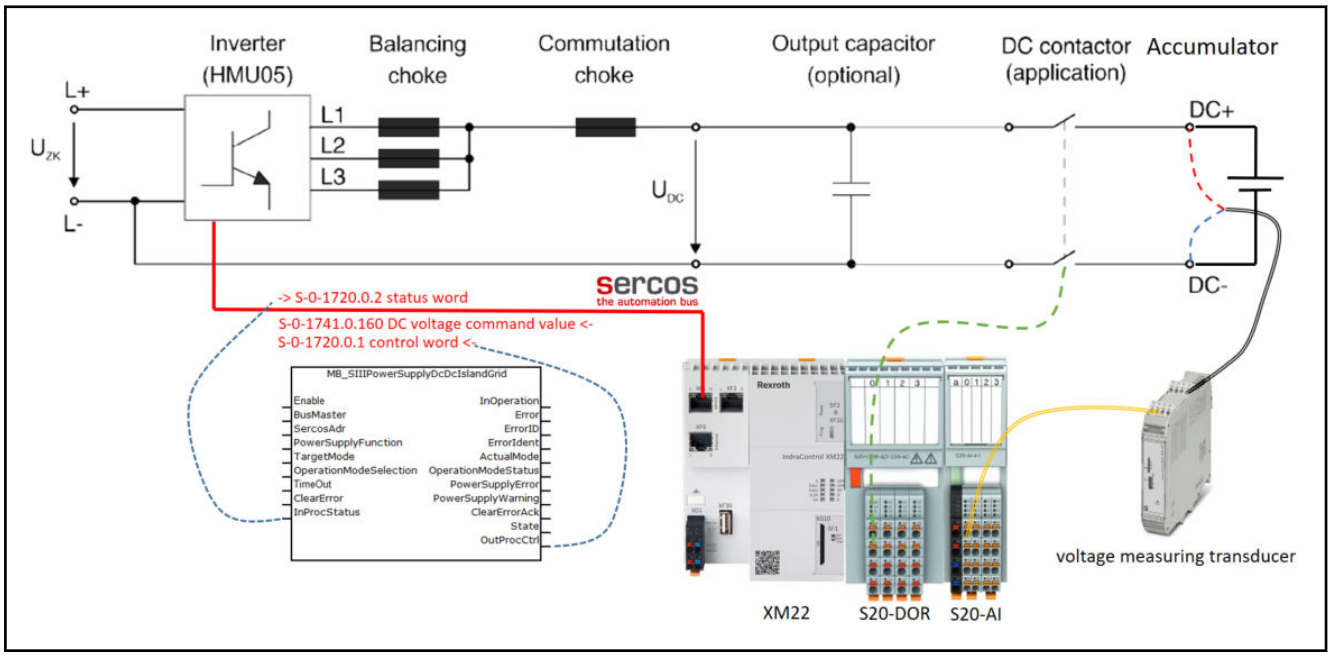


Fig. 12-28: Use case: Structural setup to charge a battery

The following parameters have to be configured cyclically at least for the HMU05:

- MDT: S-0-1720.0.1, "Supply unit control word"
- MDT: S-0-1741.0.160 "DC voltage command value"
- AT: S-0-1720.0.2, "Power supply status word"

Parameterize at least the following at the HMU05:

- Primary operation mode: Closed-loop DC voltage control
- Secondary mode 1: Closed-loop DC control
- In S-0-1741.0.150 " DC/DC converter control word", bit 0 = 1, i.e. building up DC output voltage is not required before Power Enabled

Switching the DC contactor is exemplarily performed using an S20 relay module "S20-DOR" locally connected to an XM22 control and the battery voltage is measured using the locally connected S20 analog module "S20-AI". Another hardware can also be used. The battery voltage can be measured using an analog input of the HMU and the value can be transferred using Sercos at the XM22.

Sequence in the following program example

Variable uiState	Description	"TargetMode" selection	Target state	
			ActualMode	Supply unit display
1	Power section selection off, DC contactor open	MAINS_OFF	MAINS_OFF	bb
2	Setting the voltage command value ($U_{Comm, DC}$) equal to the measured battery voltage and selecting the “Voltage control” primary mode. Power section selection off. Supply unit checks whether U_{IC} > $U_{Comm, DC}$ * 1.01? No DC output voltage yet	POWER_OFF	DC_POWER_OFF	Bb
3	Selecting the build-up of the DC output voltage	OPERATION_MODE	OPERATION_MODE	LF
4	If the DC output voltage is built-up (U_{DC} = battery voltage) and if the primary mode is acknowledged, "Bb" is specified again.	POWER_OFF	The DC output voltage should remain. Continue quickly with 5	
5	Closing the DC contactor If the DC contactor is closed, select secondary mode 1: Current control und "LF" specification	OPERATION_MODE	OPERATION_MODE	LF
6	If secondary mode 1 and "LF" are acknowledged, proceed with step "7"	OPERATION_MODE	OPERATION_MODE	LF
7	Ready. To switch off, set the variable uiState to "1".	OPERATION_MODE	OPERATION_MODE	LF

Tab. 12-45: Description of states

The procedure is controlled using the variable "uiState":

- uiState = "1" enables the function block and switches off
- uiState = "2" also enables the function block and starts the automatic procedure



In this example, the closed-loop control is disabled in step 4 and enabled again in step 5 when closing the contactor using the closed-loop current control. Depending on the application, specify the command value 0 and subsequently enable the closed-loop current control and close the contactor.

Variable definition

```
fbPowerSupplyDcDcIslandGrid: MB_SIIIPowerSupplyDcDcIslandGrid;
wPS_Status : WORD;      // connect to S-0-1720.0.2 "Power supply status word"
wPS_Control: WORD;      // connect to S-0-1720.0.1 "Power supply control word"
uiState    : UINT := 0; // set to 1 to Enable FB and stay in OFF or set to 2 for automatic start-up
```

Called cyclically

```
//-----------------------------------------------------------------------------------------------
//------------------------------ handle power-supply  -------------------------------------------
//-----------------------------------------------------------------------------------------------
CASE uiState OF
  1:  fbPowerSupplyDcDcIslandGrid.Enable := TRUE;
      // to do: Open contactor between output DC/DC converter and battery (S20-DOR)
      fbPowerSupplyDcDcIslandGrid.TargetMode := MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID.MAINS_OFF; // all OFF

  2: fbPowerSupplyDcDcIslandGrid.Enable := TRUE;
     // to do: Open contactor between output DC/DC converter and battery (S20-DOR)
     // to do: Set S-0-1741.0.160 "DC voltage command value" to measured battery voltage (S20-AI)
     // Main operating mode Voltage control
     fbPowerSupplyDcDcIslandGrid.OperationModeSelection := MB_POWERSUPPLY_MODE.OPERATIONMODE1;
     // start automatic start-up. Switch to ready for operation Bb
     fbPowerSupplyDcDcIslandGrid.TargetMode := MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID.POWER_OFF;
     IF fbPowerSupplyDcDcIslandGrid.ActualMode = MB_POWERSUPPLY_ACTUAL_MODE.DC_POWER_OFF AND
         TRUE (* to do: contactor is open? *)
     THEN
       uiState := uiState + 1;
     END_IF

  3: // Charging to LF
     fbPowerSupplyDcDcIslandGrid.TargetMode := MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID.OPERATION_MODE;
     uiState := uiState + 1;

  4: // Charged to LF AND Ack Main op-mode
     IF fbPowerSupplyDcDcIslandGrid.ActualMode = MB_POWERSUPPLY_ACTUAL_MODE.OPERATION_MODE AND
       fbPowerSupplyDcDcIslandGrid.OperationModeStatus = MB_POWERSUPPLY_MODE.OPERATIONMODE1
     THEN
       // Back to "ready for operation" Bb
       fbPowerSupplyDcDcIslandGrid.TargetMode := MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID.POWER_OFF;
       uiState := uiState + 1;
     END_IF

  5:
    // to do: Close contactor between output DC/DC converter and battery (S20-DOR)
    IF TRUE (* to do: contactor closed? *) THEN
      // activate secondary operation mode 1, (DC) current control
      fbPowerSupplyDcDcIslandGrid.OperationModeSelection := MB_POWERSUPPLY_MODE.OPERATIONMODE2;
      // Switch to operating LF
      fbPowerSupplyDcDcIslandGrid.TargetMode := MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID.OPERATION_MODE;
      uiState := uiState + 1;
    END_IF

  6: // LF reached AND Ack 2nd op-mode
     IF fbPowerSupplyDcDcIslandGrid.ActualMode = MB_POWERSUPPLY_ACTUAL_MODE.OPERATION_MODE AND
       fbPowerSupplyDcDcIslandGrid.OperationModeStatus = MB_POWERSUPPLY_MODE.OPERATIONMODE2
     THEN
       uiState := uiState + 1;
     END_IF

  7: ; // Finished. Set uiState to 1 to switch power supply OFF

  99: ; // Error!! Add error-handling here
      // to do: Open contactor between output DC/DC converter and battery (S20-DOR)

END_CASE
// call function block. I/Os are handled above
fbPowerSupplyDcDcIslandGrid(
```

```
    Enable:= ,
    InOperation=> ,
    Error=> ,
    ErrorID=> ,
    ErrorIdent=> ,
    BusMaster:= ,
    SercosAdr:= 42,
    PowerSupplyFunction:= MB_POWERSUPPLY_FUNCTION.PS_FUNCTION_DC_DC,
    TargetMode:= ,
    OperationModeSelection:= ,
    Timeout:= ,
    ClearError:= ,
    InProcStatus:= wPS_Status,
    ActualMode=> ,
    OperationModeStatus=> ,
    PowerSupplyError=> ,
    PowerSupplyWarning=> ,
    ClearErrorAck=> ,
    State=> ,
    OutProcCtrl=> wPS_Control);
IF fbPowerSupplyDcDcIslandGrid.Error THEN
  uiState := 99;
END_IF
```

12.3.4 MB_POWERSUPPLY_MODE

Brief description This enumeration is used by the "OperationModeSelection" function block input and the "OperationModeStatus" function block output.

Each enumeration element corresponds to one of the four operation modes of the power supply.

Enumeration elements

Element	Description
OPERATIONMODE1	Operation mode 1 (S-0-1709.0.1, Primary operation mode 3)
OPERATIONMODE2	Operation mode 2 (S-0-1709.0.2, Secondary operation mode 1)
OPERATIONMODE3	Operation mode 3 (S-0-1709.0.3 Secondary operation mode 2)
OPERATIONMODE4	Operation mode 4 (S-0-1709.0.4, Secondary operation mode 3)

Tab. 12-46: Elements of the MB_POWERSUPPLY_MODE enumeration

12.3.5 MB_POWERSUPPLY_ACTUAL_MODE

Brief description This enumeration is used by the "ActualMode" function block input.

Each enumeration element represents a state in the state machine of the supply unit.

Enumeration elements

Element	Description
MAINS_OFF	The supply unit is not connected to the network. If there is an output to control the contactor, the contactor is disabled.
DC_POWER_OFF	The contactor is activated. DC charging has not yet been started.
DC_BUS_CHARGING	The DC bus capacity is currently being charged This process can, if required, be interrupted by a change of the "TargetMode" input.
DC_DISCHARGING	The DC bus capacity is discharged to a lower voltage level This process can, if required, be interrupted by a change of the "TargetMode" input.
DC_BUS_CHARGED	DC bus is charged to peak value of the network The operating mode is not active
AC_POWER_OFF	The contactor is activated. Transformer magnetization and synchronization have not been started if applicable. This status is only available for the microgrid mode.

Element	Description
AC_POWER_STARTING	Transformer magnetization and synchronization are running if applicable. This status is only available for the microgrid mode.
AC_POWER_READY	Ready. Target command value reached and synchronous if applicable. This status is only available for the microgrid mode.
OPERATION_MODE	The selected operating mode is active.
ERROR	The supply unit detected an internal error and is in error state
UNKNOWN_STATE	Status word of the supply unit does not corresponds to any valid state

Tab. 12-47: MB_POWERSUPPLY_ACTUAL_MODE enumeration elements

☞ Contradictory input combinations between "ErrorMode" and "TargetMode" can occur.

In these cases, "TargetMode" dominates, irrespective of the value of the "ErrorMode" input. See the following example.

- ActualMode = _ERROR
- TargetMode = DC_POWER_OFF_NO_DISCHARGE
- ErrorMode = ERROR_TRANSITION_DISCHARGE

If the power supply leaves the "Error" state, "TargetMode" requires for the power supply to go to state "DC Power OFF" without discharging the DC bus.

"ErrorMode" requires the power supply to discharge the DC bus before switching to the next state.

In this case, "TargetMode" dominates and the power supply is switched to mode "DC Power Off" without discharging the DC bus.

This has to be taken into consideration by the programmer when programming the application.

12.3.6 MB_POWERSUPPLY_ERROR_MODE

Brief description The enumeration is used by the "ErrorMode" function block input.

Each enumeration element decides to which state the power supply switches when leaving its internal error state.

Enumeration elements

Element	Description
ERROR_TRANSITION_NO_DISCHARGE	The function block either switches to "Mains off" or to DC power off. That depends on the target mode. DC is not discharged
ERROR_TRANSITION_DISCHARGE	The function block changes to DC discharge and completes the discharge before switching to another state

Tab. 12-48: Elements of the MB_POWERSUPPLY_ERROR_MODE enumeration

☞ Contradictory input combinations between "ErrorMode" and "TargetMode" can occur.

In these cases, "TargetMode" dominates, irrespective of the value of the "ErrorMode" input. See the following example.

- ActualMode = _ERROR
- TargetMode = DC_POWER_OFF_NO_DISCHARGE
- ErrorMode = ERROR_TRANSITION_DISCHARGE

If the power supply leaves the "Error" state, "TargetMode" requires for the power supply to go to state "DC Power OFF" without discharging the DC bus.

"ErrorMode" requires the power supply to discharge the DC bus before switching to the next state.

In this case, "TargetMode" dominates and the power supply is switched to mode "DC Power Off" without discharging the DC bus.

This has to be taken into consideration by the programmer when programming the application.

12.3.7 MB_POWERSUPPLY_TARGET_MODE

Brief description The enumeration is used by the "TargetMode" function block input.

Each element of this enumeration corresponds to one of the six target modes available.

The target mode decides on the final state to be reached by the power supply as well as on the path to be taken to reach the final state.

Enumeration elements

Element	Description
MAINS_OFF_NO_DISCHARGE	The power supply has to immediately switch to state "Mains off"
MAINS_OFF_DISCHARGE	The power supply has to discharge the DC bus and immediately switch to the "Mains off" state
DC_POWER_OFF_NO_DISCHARGE	The power supply has to switch to the DC Power Off state, irrespective of the current "ActualMode" state
DC_POWER_OFF_DISCHARGE	The power supply has to discharge the bus and immediately switch to the "DC Power Off" state
DC_BUS_CHARGED	The power supply has to switch to the charged DC bus state
OPERATION_MODE	The power supply has to switch to "Operation_Mode"

Tab. 12-49: MB_POWERSUPPLY_TARGET_MODE enumeration elements

Contradictory input combinations between "ErrorMode" and "TargetMode" can occur.

In these cases, "TargetMode" dominates, irrespective of the value of the "ErrorMode" input. See the following example.

- ActualMode = _ERROR
- TargetMode = DC_POWER_OFF_NO_DISCHARGE
- ErrorMode = ERROR_TRANSITION_DISCHARGE

If the power supply leaves the "Error" state, "TargetMode" requires for the power supply to go to state "DC Power OFF" without discharging the DC bus.

"ErrorMode" requires the power supply to discharge the DC bus before switching to the next state.

In this case, "TargetMode" dominates and the power supply is switched to mode "DC Power Off" without discharging the DC bus.

This has to be taken into consideration by the programmer when programming the application.

12.3.8 MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID

Brief description

This enumeration is used by the "TargetMode" input of the "MB_SIIIPowerSupplyDcDcIslandGrid" function block.

Each element of this enumeration corresponds to one of the available target modes.

The target mode decides on the final state of the power supplier to be reached.

Enumeration elements

Element	Description
MAINS_OFF	The supply unit has to immediately switch to state "Mains off"
POWER_OFF	The supply unit has to switch to the Power Off state, irrespective of the current "ActualMode" state. Either DC_POWER_OFF or AC_POWER_OFF depending on the selected "PowerSupplyFunction"
POWER_READY	The supply unit has to switch to the Ready state. Either DC_BUS_CHARGED or AC_POWER_READY depending on the selected "PowerSupplyFunction"
OPERATION_MODE	The supply unit has to switch to "Operation_Mode"

Tab. 12-50: Elements of the enumeration MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID

12.3.9 MB_POWERSUPPLY_FUNCTION

Brief description

The enumeration is used by the "PowerSupplyFunction" function block input.

Each element of this enumeration corresponds to a function package or an extended supply unit function.

Depending on the supply unit function used, the "ActualMode" output reports other status values.

Enumeration elements

Element	Description
PS_FUNCTION_DC_DC	DC-DC converter mode, function package **DCE**
PS_FUNCTION_ISLAND_GRID	Microgrid mode, function package **MSE** -> "grid function".

Tab. 12-51: Elements of the MB_POWERSUPPLY_FUNCTION enumeration

12.4 RexrothEFC

12.4.1 MB_SIIIEfcCodeToIdn

Brief description

The MB_SIIIEfcCodeToIdn function converts a function code/parameter code of the EFC into the corresponding Sercos parameter IDN. This IDN and the function blocks to read/write parameters from the RIL_SercosIII library can be used to access the EFC parameters via Sercos.

Target system	Library
MLC/MTX 14V10 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for 20VRS and higher	RMB_SercosIII_Util

Tab. 12-52: Reference table

Interface description

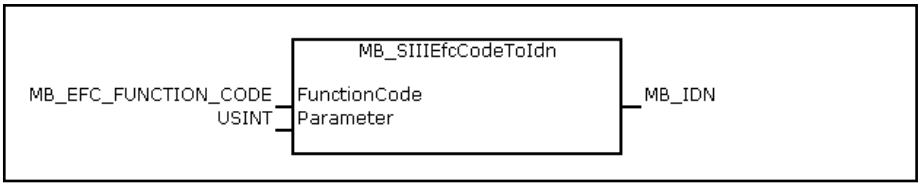


Fig. 12-29: MB_SIIIEfcCodeToIdn function

I/O type	Name	Data type	Description
VAR_INPUT	FunctionCode	MB_EFC_FUNC-TION_CODE	Function code of the parameter of the EFC, e.g. H0 at H0.10 MB_EFC_FUNCTION_CODE, page 481
	Parameters	USINT	Parameter code of the parameters of the EFC, e.g. 10 at H0.10
Return value		MB_IDN	Sercos IDN, e.g. P-0-1098.0.10 at H0.10

Example on the data type UDEC (decimal with sign)

In the following example, the parameter d0.11 "Output current" is read.

Variable definition

```
VAR
    fbSIIIReadRealParameter:    IL_SIIIReadRealParameter;
    rValue: REAL;
END_VAR
```

Reading program code d0.11 (P-0-1058.0.11)

```
fbSIIIReadRealParameter.Execute:=   TRUE;
fbSIIIReadRealParameter.SercosAdr:= 1;
fbSIIIReadRealParameter.Idn:= MB_SIIIEfcCodeToIDN(MB_EFC_FUNCTION_CODE.d0,11);

fbSIIIReadRealParameter();

IF fbSIIIReadRealParameter.Done = TRUE THEN
  rValue := fbSIIIReadRealParameter.Value;
END_IF
```

After successful reading, "0.5" is written in **rValue** for example.

12.4.2 MB_SIIIPlcDriveEfcFrequencyType01

Brief description

The function block MB_SIIIPlcDriveEfcFrequencyType01 is used to operate a Rexroth frequency converter EFC in the operation mode "Frequency control".

The function block "MB_SIIIEfcFrequencyPlcDrive" was renamed to " MB_SIIIPlcDriveEfcFrequencyType01" in 14V16 and the interface was optimized.

An "AxisInterface" example program is available as GAT plugin "SercosPlcDriveInterface" for this function block. This example program can also be used without GAT. Refer to the plugin documentation.

For the Rexroth frequency converter EFC, the parameters are labeled with function codes, e.g. b0.00. These parameters can only be accessed via the Sercos interface using the P-parameters.

Only the function codes are documented. The P-parameters are not documented.

To access the parameters of the EFC, use the function MB_SIIIEfcCodeToIdn on page 476 together with the function blocks to read and write parameters from the RIL_SercosIII library.

Target system	Library
MLC/MTX 14V10 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for 20VRS and higher	RMB_SercosIII_Util

Tab. 12-53: Reference table

Interface description

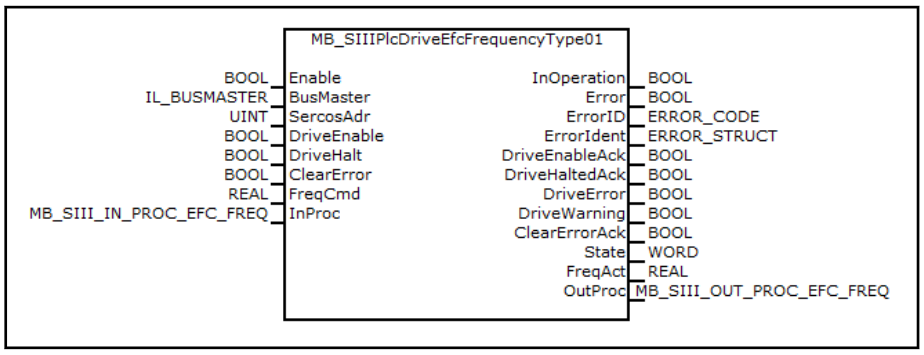


Fig. 12-30: MB_SIIIPlcDriveEfcFrequencyType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device.

I/O type	Name	Data type	Description
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	FreqCmd	REAL	Frequency command value. Unit in [Hz]. For the reverse operation, specify a negative value.
	InProc	MB_SIII_IN_PROC_EFC_FREQ	Applying process input data %I. MB_SIII_IN_PROC_EFC_FREQ on page 481
VAR_OUT-PUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 480
	DriveEnableAck	BOOL	Drive enabled acknowledged, S-0-0135, Drive status word, bit 14/15
	DriveHaltedAck	BOOL	Drive halt acknowledged, S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error, S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning, S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed, S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Current drive status. See below.
	FreqAct	REAL	Current output frequency. Unit in [Hz]. In case of reverse operation, negative values are output.
	OutProc	MB_SIII_OUT_PROC_EFC_FREQ	Applying process output data %I. MB_SIII_OUT_PROC_EFC_FREQ on page 481

Tab. 12-54: MB_SIIIPlcDriveEfcFrequencyType01 interface

"State" output The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error

Bit no.	Name	Description
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Bus master	IL_BUSMASTER			0	Rising edge at “Enable”
SercosAdr	UINT	0 (MLD: local axis) 1 (other systems)	System-specific	0	Continuous
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
FreqCmd	REAL	-400,0 (defined by E0.08 and E0.09)	+400,0 (defined by E0.08 and E0.09)	0.0	Continuous
InProc	MB_SIII_IN_PROC_EFC_FREQ			0	Continuous

Tab. 12-55: Min./max. and default values of the inputs MB_SIIIPlcDriveEfcFrequencyType01

Functional description

The MB_SIIIPlcDriveEfcFrequencyType01 function block extends the MB_SIIIPlcDriveType01 function block and is provided with all its inputs and outputs as well as its functionality.

Additionally, it is provided with inputs and outputs:

- For the actual and command values of this operation mode:
  - The “FreqCmd” input indicates the frequency command value of the axis. The value is cyclically transferred to the drive via the "OutProc.Frequency" output and the parameter P-0-1098.0.10 (H0.10). The value is limited to a maximum value of P-0-1074.0.8 (E0.08) "Maximum output frequency".
  - The "FreqAct" output outputs the actual frequency value of the axis. The value is cyclically read by the drive via the parameter P-0-1058.0.10 (d0.00) and the input "InProc.Frequency".
- For additional special status words and control words for the EFC:
  - The "InProc.EfcStatus" input for status messages of the EFC, e.g., deceleration, acceleration and reverse operation can be retrieved using this input.
  - The "OutProc.EfcCtrl" output to control the EFC (e.g. forward/backward switching).

Configuring cyclic data in Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Ex-

plorer below the Sercos node. Therefore, select the “Drive” node below the respective device.

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

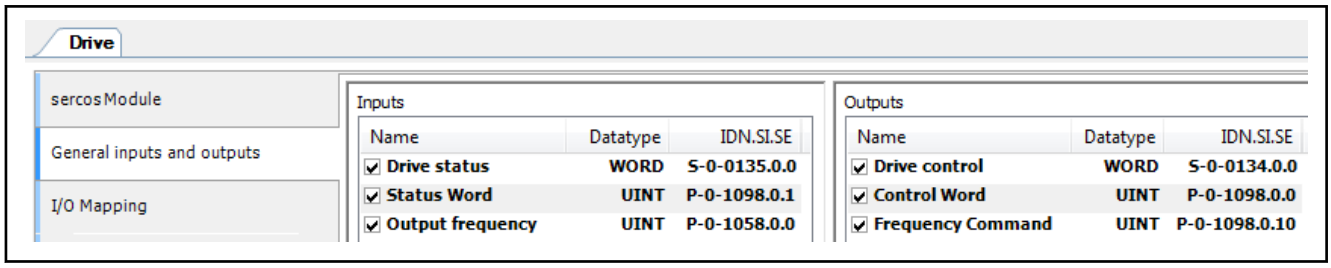


Fig. 12-31: Configuring cyclic parameters

Subsequently, the corresponding input variables and output variables of the function block are assigned in the "I/O mapping" tab.

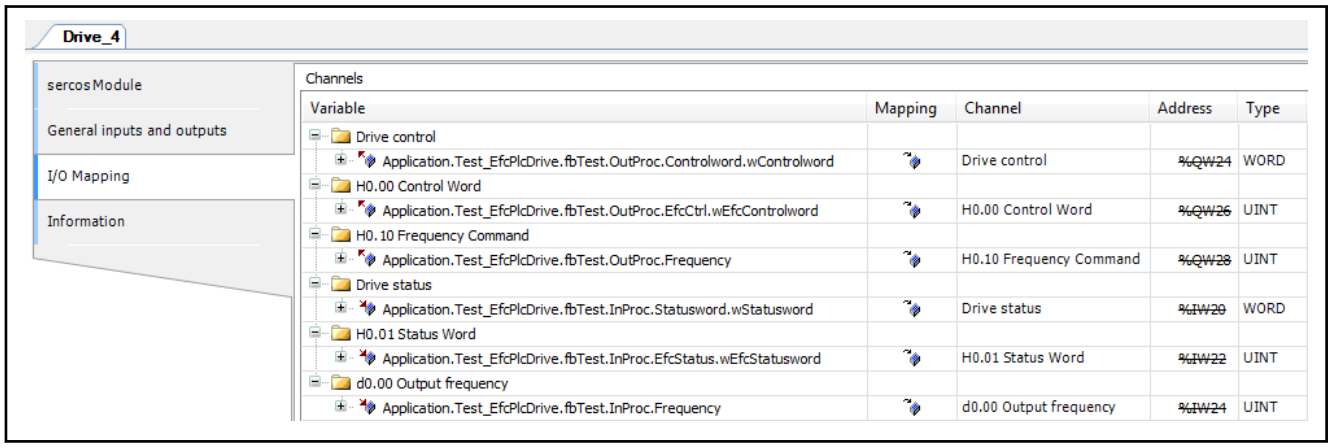


Fig. 12-32: Mapping input variables and output variables

Parameterizing the drive:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The MB_SIIIPlcDriveEfcFrequencyType01 requires the parameterization of the main operation mode S-0-0032 to "frequency control" (value 2#1000.0000.0000.0000). For the further parameterization for the "frequency control" mode, use IndraWorks Ds or ConverterWorks for example.

A basic parameterization of the Rexroth frequency converter (limit values, etc.) is required in any case.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#00002985	Attribute value	Attribute of the parameter P-0-1098.0.10 or P-0-1058.0.0 is not two bytes decimal without sign.

Tab. 12-56: Error codes of the MB_SIIIPlcDriveEfcFrequencyType01 function block

12.4.3 MB_EFC_FUNCTION_CODE

Brief description The enumeration type MB_EFC_FUNCTION_CODE describes the function code of the EFC.

Name	Value	Description
b0..b9	16#00..09	Function code of the system parameters b0 to b9
d0..d9	16#10..19	Function code of the monitoring parameters d0 to d9
C0..C9	16#20..29	Function code of the performance parameters C0 to C9
E0..E9	16#30..39	Function code of the function control parameters E0 to E9
U0..U9	16#40..49	Function code of the operating panel parameters U0 to U9
F0..F9	16#50..59	Function code of the APP parameters F0 to F9
H0..H9	16#60..69	Function code of the option card parameters H0 to H9

Tab. 12-57: Elements of the enumeration type MB_EFC_FUNCTION_CODE

12.4.4 MB_SIII_IN_PROC_EFC_FREQ

Brief description The data structure MB_SIII_IN_PROC_EFC_FREQ contains the process input data of the function block MB_SIIIPlcDriveEfcFrequencyType01 on page 477.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135 to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
EfcStatus	MB_SIII_EFC_STATUSWORD		Assign input variable %IW of the status word P-0-1098.0.1 to the "wEfcStatusword" element. MB_SIII_EFC_STATUSWORD on page 482
Frequency	UINT	16#0	Assign input variable %IW to the current output frequency P-0-1058.0.0 (d0.00).

Tab. 12-58: Elements of the MB_SIII_IN_PROC_EFC_FREQ structure

12.4.5 MB_SIII_OUT_PROC_EFC_FREQ

Brief description The data structure MB_SIII_OUT_PROC_EFC_FREQ contains the process output data of the function block MB_SIIIPlcDriveEfcFrequencyType01 on page 477.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134 to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
EfcCtrl	MB_SIII_EFC_CONTROLWORD		Assign output variable %QW of the control word P-0-1098 to the "wControlword" element. MB_SIII_EFC_CONTROLWORD on page 482
Frequency	UINT	16#0	Assign output variable %QW of the frequency command P-0-1098.0.10 (H0.10).

Tab. 12-59: Elements of the MB_SIII_OUT_PROC_EFC_FREQ structure

12.4.6 MB_SIII_EFC_CONTROLWORD

Brief description

The data structure of the MB_SIII_EFC_CONTROLWORD corresponds to the parameter P-0-1098, H0.00 "Tab for the communication control" of the EFC. The bits can only be read in the user program.

The variable "wEfcControlword" is stored in the same memory as the bits and is used to map in the I/O mapping of the EFC.

Structural element	Type	Min. value	Max. value	Default value	Description
wEfcControlword	WORD			0	Complete check word
RunCommand	BIT			FALSE	1= Run command active
Jogging	BIT			FALSE	1= Jog mode active (step direction determined by bit 2)
Reverse	BIT			FALSE	1= Backward, 0 = Forward
Stop	BIT			FALSE	1= Stop acc. to parameter setting
EStop	BIT			FALSE	1= E-STOP active
FaultReset	BIT			FALSE	1= Resetting error active
StopAccDec	BIT			FALSE	1= Stop acc./dec. active (stops the internal acceleration ramp generator/deceleration jump generator)
ControlWordActive	BIT			FALSE	1= Control word active

Tab. 12-60: Elements of the MB_SIII_EFC_CONTROLWORD structure

12.4.7 MB_SIII_EFC_STATUSWORD

Brief description

The data structure MB_SIII_EFC_STATUSWORD corresponds to the parameter P-0-1098.0.1, H0.01 "Tab for feedback of the communication status" of the EFC. The data can only be read in the user program.

The variable "wEfcStatusword" is stored in the same memory as the bits and is used to map in the I/O mapping of the EFC.

Structural element	Type	Min. value	Max. value	Default value	Description
wEfcStatusword	WORD			0	Complete status word
Reverse	BIT			FALSE	1= Backward, 0= Forward
Running	BIT			FALSE	1= Start, 0= Stop

Structural element	Type	Min. value	Max. value	Default value	Description
Jogging	BIT			FALSE	1= Jog mode
Accelerating	BIT			FALSE	1= Acceleration
Decelerating	BIT			FALSE	1= Deceleration
Voltage	BIT			FALSE	1= Limitation overvoltage, 0= Normal
Current	BIT			FALSE	1= Limitation overcurrent, 0= Normal
Error	BIT			FALSE	1= Errors
ErrorCode	BYTE			16#00	Error code (corresponds to E9.05)

Tab. 12-61: Elements of the MB_SIII_EFC_STATUSWORD structure

12.5 RexrothHydraulicDrives

12.5.1 MB_SIIIPlcDriveFlowControl

Brief description

The function block MB_SIIIPlcDriveFlowControl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Flow rate control**.



For the details to possible operation modes, see:

Rexroth HydraulicDrive HDx-20 Functions, Application Description, DOK-HYDRV*-HD*-20VRS**-APxx-EN-P, RD-30338-FK, chapter 6, Operation modes

The operation mode is enabled and disabled in the device. A flow rate is specified as command value with a set scaling. The actual flow rate value and the effective valve command value as well as the status messages are displayed.

Target system	Library
MLC/MTX ≥ 14V16	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ HDx 20VRS or MPx 20VRS	RMB_SercosIII_Util

Tab. 12-62: Reference table

Interface description

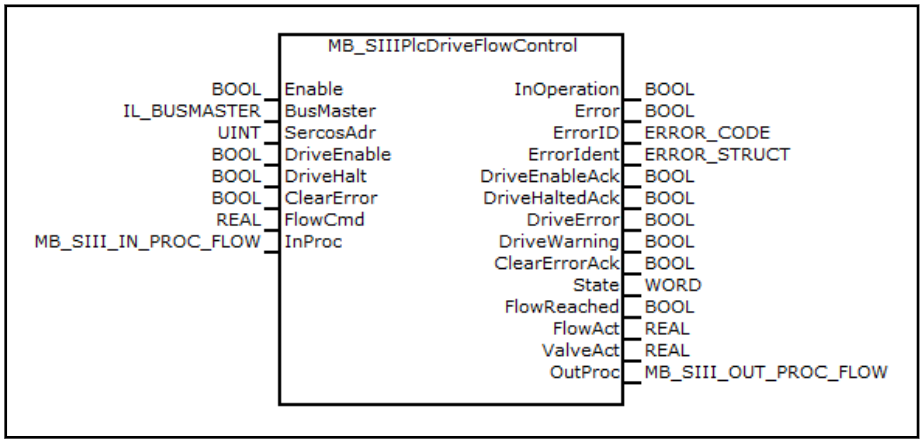


Fig. 12-33: MB_SIIIPlcDriveFlowControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “Bus-Master” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	FlowCmd	REAL	S-0-0840, Flow command value, scaling acc. to attribute
	InProc	MB_SIII_IN_PROC_FLOW	Applying process input data %I. MB_SIII_IN_PROC_FLOW, page 526
VAR_OUTPUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 487
	DriveEnableAck	BOOL	Drive enabled acknowledged.
	DriveHaltedAck	BOOL	Drive halt acknowledged. S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error. S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning. S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed. S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Device status of IL_RemoteSlaveState
	FlowReached	BOOL	S-0-0849, Flow status word, bit 1
	FlowAct	REAL	S-0-0842, Flow feedback value, scaling acc. to attribute

I/O type	Name	Data type	Description
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	OutProc	MB_SIII_OUT_PROC_FLOW	Apply process output data %Q. MB_SIII_OUT_PROC_FLOW, page 527

Tab. 12-63: MB_SIIIPlcDriveFlowControl interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
FlowCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_FLOW				Continuous

Functional description

The function block switches the drive to the operation mode "Flow rate control" (operation mode code: 0x0052). A flow rate command value is specified at the "FlowCmd" input. The actual value of the flow rate is displayed at the "FlowAct" output. These floating point numbers (REAL) are internally converted into fixed point values of the device (DINT) using the structures "InProc" and "OutProc".

Reaching of the flow rate is displayed at the output “FlowReached” (S-0-0849, Flow status word, bit 1).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes. A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Ex-

plorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0144, Signal status word
- S-0-0842, Flow feedback value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0840, Flow command value

At first, configure the cyclic parameters in the “General inputs and outputs” tab.

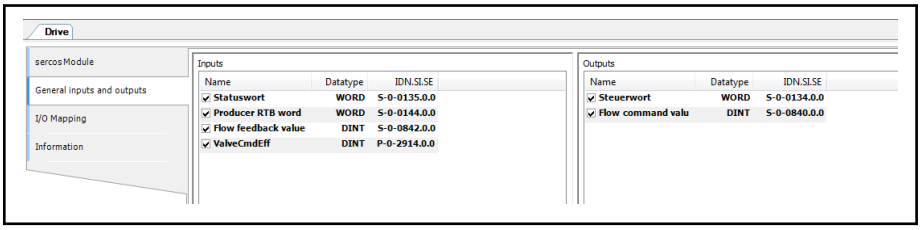


Fig. 12-34: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, it is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

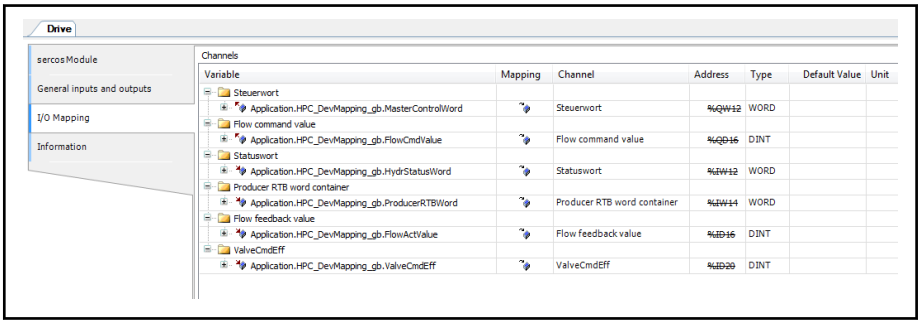


Fig. 12-35: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- “Flow rate control” (16#0052) has to be entered into the parameter S-0-0032 Primary operation mode. The main operation mode is selected.

- The parameter S-0-0849 Flow status word, bit number 1 ("flow rate command = actual flow rate") has to be entered as bit 1 into the parameter S-0-0144 Signal status word.

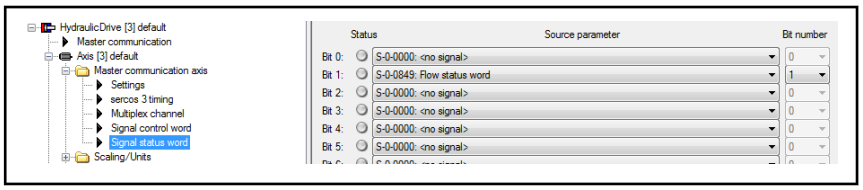


Fig. 12-36: Parameterizing the signal status word S-0-0144

- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B0	Attribute value	Attribute of the parameters S-0-0840 or P-0-2914 is not four bytes decimal with sign

Tab. 12-64: Error codes of the MB_SIIIPlcDriveFlowControl function block

12.5.2 MB_SIIIPlcDrivePressureFlowControl

Brief description

The function block MB_SIIIPlcDrivePressureFlowControl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Pressure and flow rate control**.

For the details to possible operation modes, see:

Rexroth HydraulicDrive HDx-20 Functions, Application Description, DOK-HYDRV*-HD*-20VRS**-APxx-EN-P, RD-30338-FK, chapter 6, Operation modes

The operation mode is enabled and disabled in the device. A flow rate and a pressure value are specified as command value with the respective scaling. The actual pressure valve value, the actual flow rate value and the effective valve command value as well as the status messages are displayed. The active closed-loop control mode is displayed at the output "ControlMode".

Target system	Library
MLC/MTX 14V16 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for HDx 20VRS or MPx 20VRS and higher	RMB_SercosIII_Util

Tab. 12-65: Reference table

Interface description

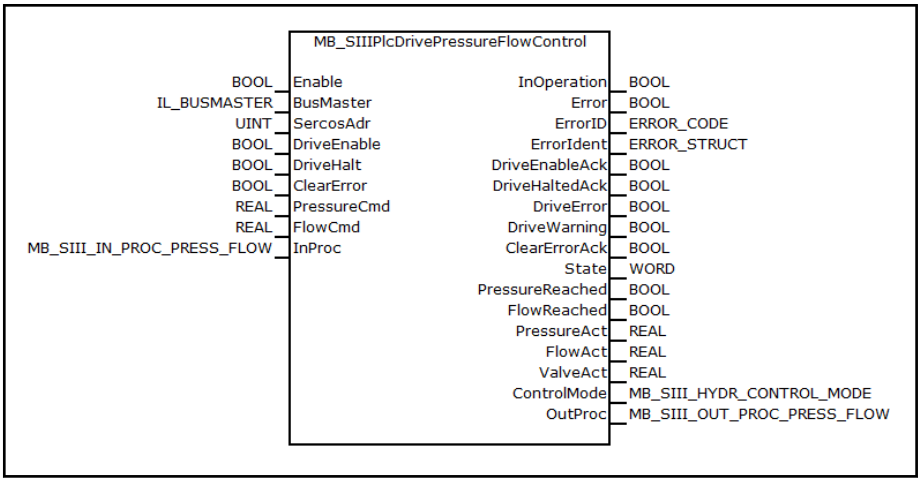


Fig. 12-37: MB_SIIIPlcDrivePressureFlowControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMAS-TER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	PressureCmd	REAL	S-0-0800, Pressure command value, scaling acc. to attribute
	FlowCmd	REAL	S-0-0840, Flow command value, scaling acc. to attribute
	InProc	MB_SIII_IN_PROC_PRES-SURE_FLOW	Applying process input data %I. MB_SIII_IN_PROC_PRESSURE_FLOW on page 527
VAR_OUT-PUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_COD E	Diagnostic description in case of error
	ErrorIdent	ERROR_STR UCT	Detailed diagnostics Troubleshooting, page 492
	DriveEnableAck	BOOL	Drive enabled acknowledged, S-0-0135, Drive status word, bit 14/15
	DriveHaltedAck	BOOL	Drive halt acknowledged, S-0-0135, Drive status word, bit 4

I/O type	Name	Data type	Description
	DriveError	BOOL	Drive error, S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning, S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed, S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Current drive status. See below.
	PressureReach-ed	BOOL	S-0-0830, Pressure status word, bit 1
	FlowReached	BOOL	S-0-0849, Flow status word, bit 1
	PressureAct	REAL	S-0-0809, Pressure feedback value, scaling acc. to attribute
	FlowAct	REAL	S-0-0842, Flow feedback value, scaling acc. to attribute
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	ControlMode	MB_SIII_HYDR_CON-TROL_MODE	P-0-2951, Pump controller status word or P-0-2801, p/Q controller status word, bit 0 (1 pressure controller, 0 flow rate controller active) MB_SIII_HYDR_CONTROL_MODE, page 526
	OutProc	MB_SIII_OUT_PROC_PRESSURE_FLOW	Applying process output data %I. MB_SIII_OUT_PROC_PRESSURE_FLOW, page 528

Tab. 12-66: MB_SIIIPlcDrivePressureFlowControl interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
PressureCmd	REAL	N. def.	N. def.	0.0	Continuous
FlowCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_PRESSURE_FLOW				Continuous

Functional description

The function block switches the drive to the operation mode "Pressure control/Flow rate control" (operation mode code: 16#0055). A pressure command value is specified at the "PressureCmd" input and a flow rate command value is specified at the "FlowCmd" input.

The actual value of the pressure is displayed at the "PressureAct" output and the actual value of the flow rate is displayed at the "FlowAct" output.

These floating point numbers (REAL) are internally converted into fixed point values of the device (DINT) using the structures "InProc" and "OutProc".

Reaching of the actual pressure value is displayed at the output "PressureReached" (S-0-0830, Pressure status word, bit 1) and reaching of the flow rate is displayed at the output "FlowReached" (S-0-0849, Flow status word, bit 1).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes.

The active closed-loop control mode is displayed at the output "ControlMode" (closed-loop pressure control or flow rate control).

A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0144, Signal status word
- S-0-0842, Flow feedback value
- S-0-0809, Pressure feedback value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0840, Flow command value
- S-0-0800, Pressure command value

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

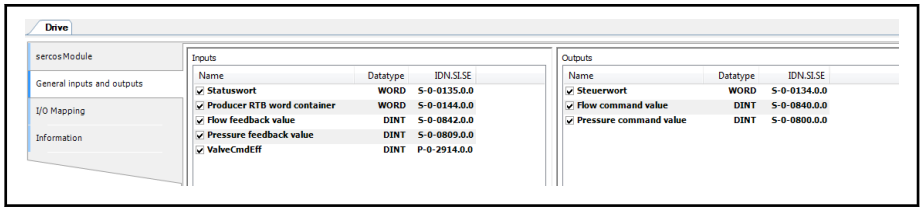


Fig. 12-38: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, it is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

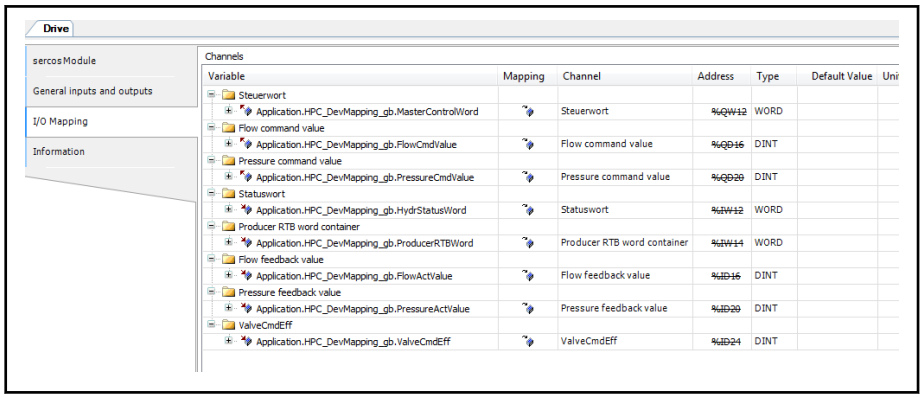


Fig. 12-39: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- "Pressure/flow rate control" (16#0055) has to be entered into the parameter S-0-0032 Primary operation mode. The main operation mode is selected.
- Setting parameters and bits, S-0-0144, Signal status word
  - Bit 0: S-0-0830, Pressure status word, bit 1
  - Bit 1: S-0-0849, Flow status word, bit 1
  - Bit 2: P-0-2951, Pump controller status word, bit 0 for the function package “Pump control”

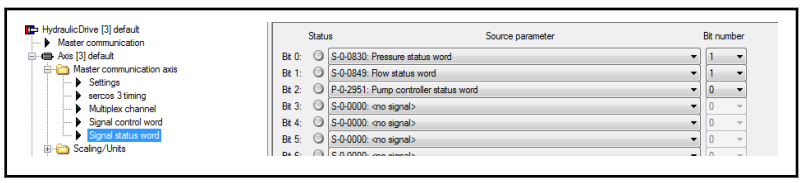


Fig. 12-40: Parameterizing the signal status word S-0-0144

  - (Bit 2: P-0-2801, p/Q controller status word, bit 0 for function package P/Q)
  - …
- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

☞ Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B1	Attribute value	Attribute of the parameters S-0-0800 or S-0-0840 or P-0-2914 is not four bytes decimal with sign

Tab. 12-67: Error codes of the MB_SIIIPlcDrivePressureFlowControl function block

12.5.3 MB_SIIIPlcDrivePressureSwivelAngleControl

Brief description The function block MB_SIIIPlcDrivePressureSwivelAngleControl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Pressure and swiveling angle control**.

☞ For the details to possible operation modes, see:

Rexroth HydraulicDrive HDx-20 Functions, Application Description, DOK-HYDRV*-HD*-20VRS**-APxx-EN-P, RD-30338-FK, chapter 6, Operation modes

The operation mode is enabled and disabled in the device. A pressure value and a swiveling value are specified as command value with the respective scaling. The actual pressure valve value, the actual swiveling angle value and the effective valve command value as well as the status messages are displayed.

The active closed-loop control mode is displayed at the output "ControlMode".

Furthermore, when enabling the function block, the master functionality is checked in the parameter P-0-2950, Pump controller control word and set if not available.

Target system	Library
MLC/MTX ≥ 14V16	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ HDx 20VRS or MPx 20VRS	RMB_SercosIII_Util

Tab. 12-68: Reference table

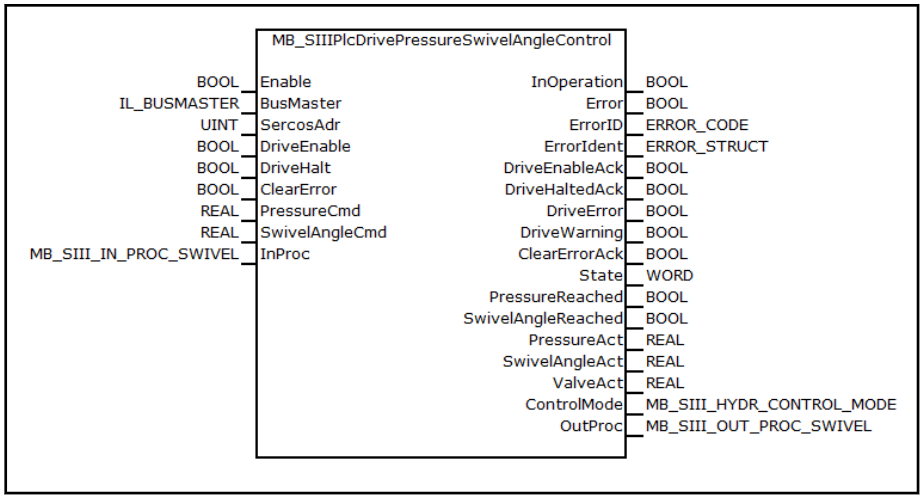


Fig. 12-41: MB_SIIIPlcDrivePressureSwivelAngleControl function block

Interface description

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMAS-TER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	PressureCmd	REAL	S-0-0800, Pressure command value, scaling acc. to attribute
	SwivelAngleCmd	REAL	S-0-0880, Swivel angle command value, scaling specified
	InProc	MB_SIII_IN_PROC_SWIV-EL	Applying process input data %I. MB_SIII_IN_PROC_SWIVEL, page 529
VAR_OUT-PUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_COD E	Diagnostic description in case of error
	ErrorIdent	ERROR_STR UCT	Detailed diagnostics Troubleshooting, page 497
	DriveEnableAck	BOOL	Drive enabled acknowledged.

I/O type	Name	Data type	Description
	DriveHaltedAck	BOOL	Drive halt acknowledged. S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error. S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning. S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed. S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Current drive status. See below.
	PressureReach-ed	BOOL	S-0-0830, Pressure status word, bit 1
	SwivelAn-gleReached	BOOL	S-0-0849, Flow status word, bit 1
	PressureAct	REAL	S-0-0809, Pressure feedback value, scaling acc. to attribute
	SwivelAngleAct	REAL	S-0-0882, Swivel angle actual value acc. to attribute
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	ControlMode	MB_SIII_HYDR_CON-TROL_MODE	P-0-2951, Pump controller status word or P-0-2801, p/Q controller status word, bit 0 (1 pressure controller, 0 flow rate controller active) MB_SIII_HYDR_CONTROL_MODE, page 526
	OutProc	MB_SIII_OUT_PROC_SWIVEL	Apply process output data %Q. MB_SIII_OUT_PROC_SWIVEL, page 529

Tab. 12-69: MB_SIIIPlcDrivePressureSwivelAngleControl interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
PressureCmd	REAL	N. def.	N. def.	0.0	Continuous
SwivelAngleCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_SWIVEL				Continuous

Functional description

The function block switches the regulating pump to the operation mode "Pressure control/Swiveling angle control" (operation mode code: 0x0056). When enabling the function block (via the "Enable" input), it is automatically checked via the parameter P-0-2950, Pump controller control word, bit 5 = 0 that the master functionality is set for the pump. A pressure command value is specified at the "PressureCmd" input and a swiveling angle command value is specified at the "SwivelAngleCmd" input.

The actual value of the pressure is displayed at the "PressureAct" output and the actual value of the swiveling angle is displayed at the "SwivelAngleAct" output.

Reaching of the actual pressure value is displayed at the output "PressureReached" (S-0-0830, Pressure status word, bit 1) and reaching of the swiveling angle value is displayed at the output "SwivelAngleReached" (S-0-0887, Swivel angle status word, bit 0).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes.

The active closed-loop control mode is displayed at the output "ControlMode" (closed-loop pressure control or swiveling angle control).

A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0144, Signal status word
- S-0-0809, Pressure feedback value
- S-0-0882, Swivel angle actual value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0800, Pressure command value
- S-0-0880, Swivel angle command value

At first, configure the cyclic parameters in the “General inputs and outputs” tab.

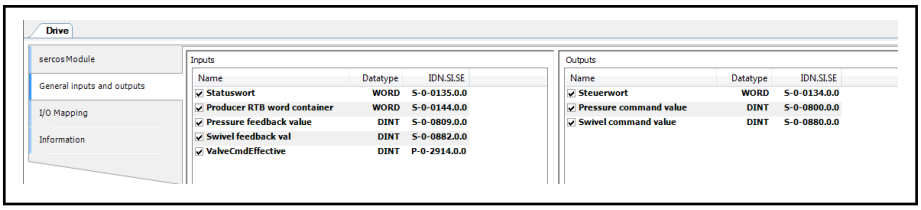


Fig. 12-42: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, it is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

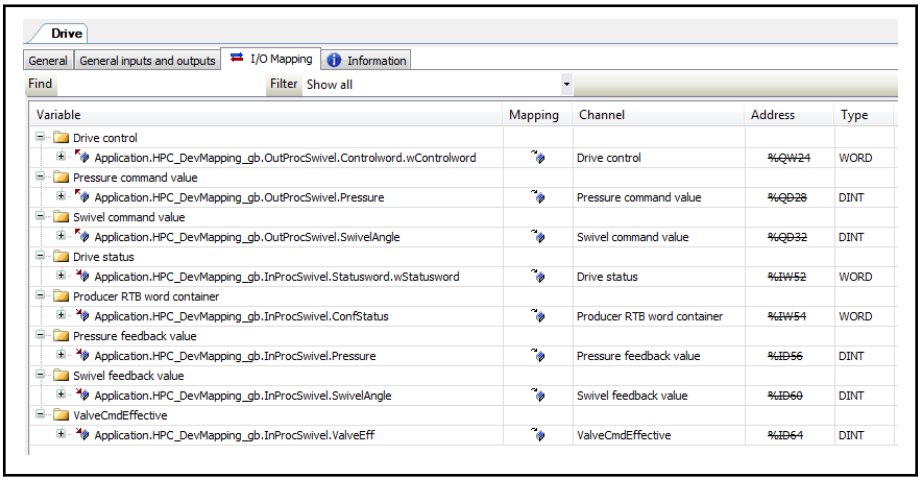


Fig. 12-43: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- "Pressure/swiveling angle control" (16#0056) has to be entered into the parameter S-0-0032 Primary operation mode. The main operation mode is selected.
- The parameter S-0-0830, Pressure status word, bit 1 ("Pressure command value reached") has to be entered as bit 0 into the parameter S-0-0144 Signal status word.
- The parameter S-0-0887, Swivel angle status word, bit 1 ("Swiveling angle command value reached") has to be entered as bit 3 into the parameter S-0-0144 Signal status word.
- The parameter P-0-2951, Pump controller status word, bit 0 ("Pressure command value reached") has to be entered as bit 4 into the parameter S-0-0144 Signal status word.

- The parameter P-0-2951, Pump controller status word, bit 1 ("Pressure command value reached") has to be entered as bit 5 into the parameter S-0-0144 Signal status word.

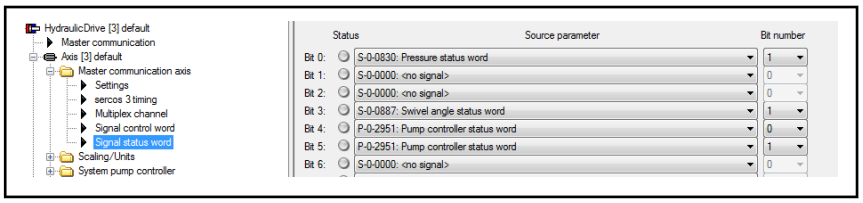


Fig. 12-44: Parameterizing the signal status word S-0-0144

- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B2	Attribute value	Attribute of the parameters S-0-0800 or S-0-0880 or P-0-2914 is not four bytes decimal with sign

Tab. 12-70: Error codes of the MB_SIIIPlcPressureSwivelAngleControl function block

12.5.4 MB_SIIIPlcDrivePressureSwivelAngleSlave

Brief description

The function block MB_SIIIPlcDrivePressureSwivelAngleSlave extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Pressure and swiveling angle control** (for slave).

For the details to possible operation modes, see:

Rexroth HydraulicDrive HDx-20 Functions, Application Description, DOK-HYDRV*-HD*-20VRS**-APxx-EN-P, RD-30338-FK, chapter 6, Operation modes

The operation mode is enabled and disabled in the device. A pressure value and a swiveling value are specified as command value with the respective scaling. The actual pressure valve value, the actual swiveling angle value and the effective valve command value as well as the status messages are displayed.

Furthermore, when enabling the function block, the slave functionality is set automatically in the parameter P-0-2950, Pump controller control word.

Target system	Library
MLC/MTX ≥ 14V16	RMB_SercosIII_Util
MLD with IndraLogic 2G ≥ HDx 20VRS or MPx 20VRS	RMB_SercosIII_Util

Tab. 12-71: Reference table

Interface description

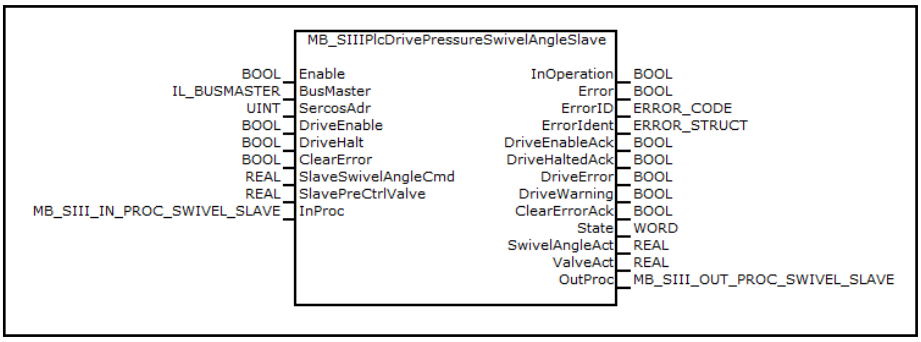


Fig. 12-45: MB_SIIIPlcDrivePressureSwivelAngleSlave function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMAS-TER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER, page 415
	SercosAdr	UINT	Sercos III address of the slave device.
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	SlaveSwivelAn-gleCmd	REAL	P-0-2935, Swivel angle command value for master/slave function, scaling specified
	SlavePreCtrl-Valve	REAL	Precontrol slave P-0-2983, Output adjustment, feedforward valve
	InProc	MB_SIII_IN_P ROC_SWIV-EL_SLAVE	Applying process input data %I. MB_SIII_IN_PROC_SWIVEL_SLAVE, page 530
VAR_OUT-PUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_COD E	Diagnostic description in case of error
	ErrorIdent	ERROR_STR UCT	Detailed diagnostics Troubleshooting, page 501

I/O type	Name	Data type	Description
	DriveEnableAck	BOOL	Drive enabled acknowledged.
	DriveHaltedAck	BOOL	Drive halt acknowledged. S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error. S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning. S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed. S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Current drive status. See below.
	SwivelAngleAct	REAL	Created from S-0-0882, Swivel angle actual value
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	OutProc	MB_SIII_OUT_PROC_SWIVEL_SLAVE	Apply process output data %Q. MB_SIII_OUT_PROC_SWIVEL_SLAVE, page 530

Tab. 12-72: MB_SIIIPlcDrivePressureSwivelAngleSlave interface

"State" output

The "State" output provides a rough diagnostic classification in binary code.

The bits are included in the RIL_FieldbusTypes.library as global "Fieldbus-States" constants.

Bit no.	Name	Description
0	IL_RBS_SLAVE_NOT_REACHABLE	Slave does not exist
1	IL_RBS_SLAVE_NOT_READY	Slave not ready
2	IL_RBS_SLAVE_CONFIG_ERROR	Slave configuration error
3	IL_RBS_SLAVE_DIAGNOSIS	Slave diagnostics available
4	IL_FBS_SLAVE_ERROR	Slave error available
5	IL_RBS_CON_OFF	At least one cyclic connection is not running

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
SlaveSwivelAngleCmd	REAL	N. def.	N. def.	0.0	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
SlavePreCtrlValve	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_SWIVEL_SLAVE				Continuous

Functional description

The function block switches the regulating pump system to the operation mode "Pressure control/Swiveling angle control" in a multi-pump (operation mode code: 0x0056). When enabling the function block (via the "Enable" input), the slave functionality is automatically set via the parameter P-0-2950, Pump controller control word, bit 5 = 1.

A swiveling angle command value is specified at the input "SwivelAngleCmd" originating from a regulating pump with master functionality. Additionally, the valve command value of the master pump can be added as precontrol word (P-0-2983, Output adjustment, feedforward valve) via the input "SlavePreCtrlValve".

Configuring cyclic data for Sercos

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0882, Swivel angle actual value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0880, Swivel angle command value
- P-0-2935, Swivel angle command value for master/slave function
- P-0-2983, Output adjustment, feedforward valve

At first, configure the cyclic parameters in the “General inputs and outputs” tab.

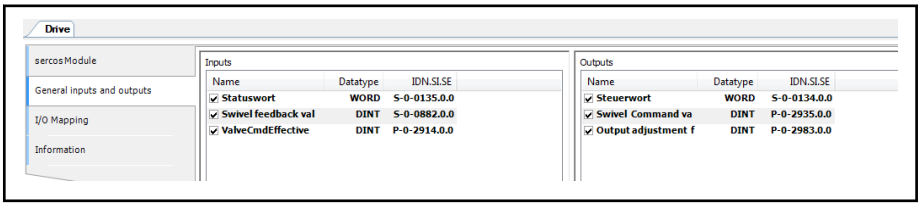


Fig. 12-46: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, it is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

Functional description

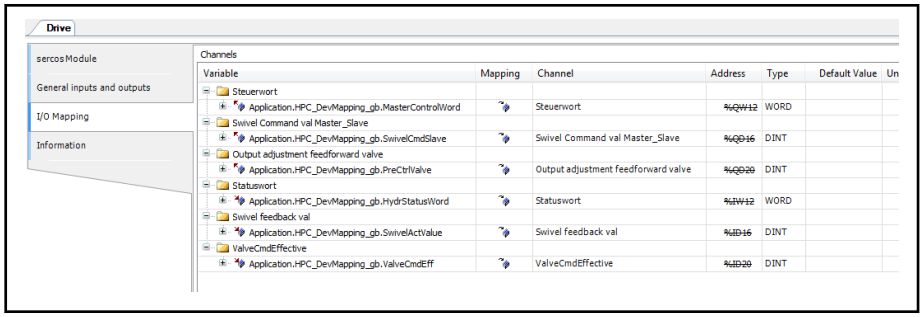


Fig. 12-47: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- "Pressure/swiveling angle control" (16#0056) has to be entered into the parameter S-0-0032 Primary operation mode. The main operation mode is selected.
- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B3	Attribute value	Attribute of the parameters P-0-2983, P-0-2935 or P-0-2914 is not four bytes decimal with sign

Tab. 12-73: Error codes of the MB_SIIIPlcDrivePressureSwivelAngleSlave function block

12.5.5 MB_SIIIPlcDriveDirectValveControl

Brief description

The function block MB_SIIIPlcDriveDirectValveControl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Direct valve control**.

Value in the operation mode parameter: 0x0051



For the details to possible operation modes, see:

Rexroth HydraulicDrive HDx-20 Functions, Application Description, DOK-HYDRV*-HD*-20VRS**-APxx-EN-P, RD-30338-FK, chapter 6, Operation modes

Target system	Library
MLC/MTX for 14V20 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for HDx 20VRS or MPx 20VRS and higher	RMB_SercosIII_Util

Tab. 12-74: Reference table

Interface description

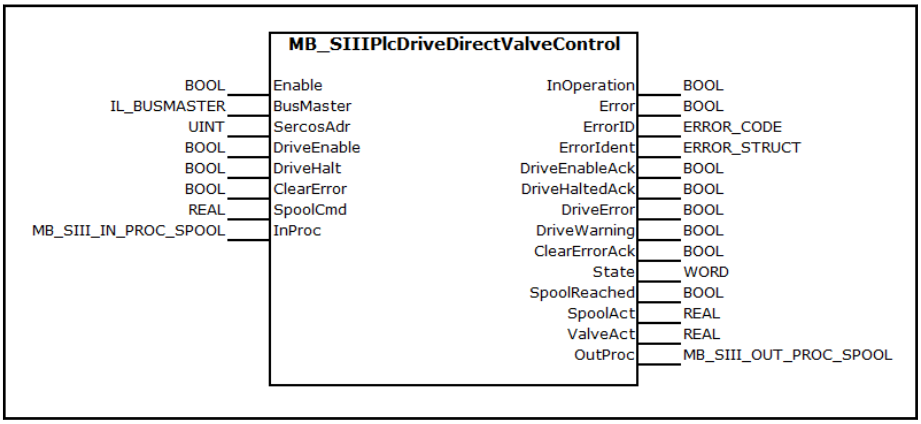


Fig. 12-48: MB_SIIIPlcDriveDirectValveControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0`. Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again.
	SpoolCmd	REAL	S-0-0860, Spool command value, scaling acc. to attribute
	InProc	MB_SIII_IN_PROC_SPOOL	Input structure MB_SIII_IN_PROC_SPOOL, page 531
VAR_OUTPUT	InOperation	BOOL	Function block is operating

I/O type	Name	Data type	Description
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 505
	DriveEnableAck	BOOL	Drive enabled acknowledged
	DriveHaltedAck	BOOL	Drive halt acknowledged S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Device status of IL_RemoteSlaveState
	SpoolReached	BOOL	Valve command value reached (S-0-0867, Spool status word bit 0)
	SpoolAct	REAL	S-0-0862, Spool feedback value, scaling acc. to attribute
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	OutProc	MB_SIII_OUT_PROC_SPOOL	Output structure MB_SIII_OUT_PROC_SPOOL, page 531

Tab. 12-75: MB_SIIIPlcDriveDirectValveControl interface

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
SpoolCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_SPOOL				Continuous

Functional description The function block switches the drive to the operation mode "Direct valve control" (operation mode code: 16#0051). A valve command value is specified at the "SpoolCmd" input.

The actual valve value is shown at the output "SpoolAct".

These floating point numbers (REAL) are internally converted into fixed point values of the device (DINT) using the structures "InProc" and "OutProc".

Reaching of the valve command value is displayed at the output "SpoolReached" (S-0-0867, Spool status word bit 0).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes.

A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0144, Signal status word
- S-0-0862, Spool feedback value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0860, Spool command value

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

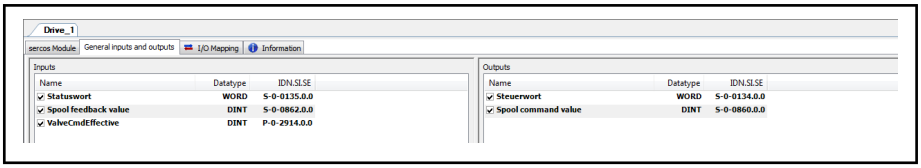


Fig. 12-49: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, this structure is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

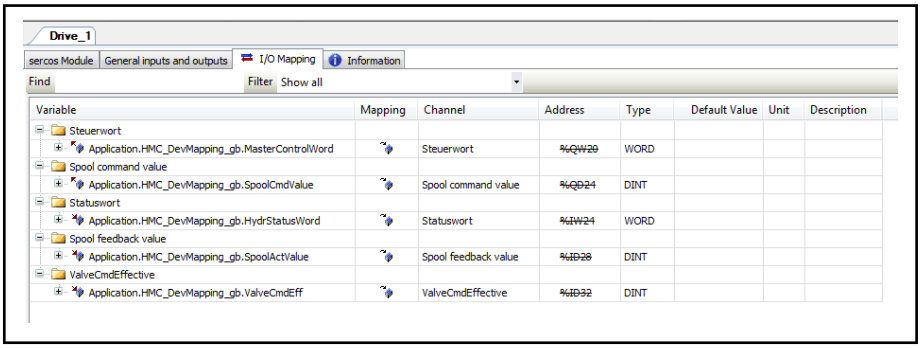


Fig. 12-50: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- Setting main operation mode to direct valve control
  - S-0-0032 " Primary operation mode" Value "0x0051"
- Setting parameters and bits, S-0-0144 " Signal status word"
  - …
  - Bit 6: S-0-0867 " Spool status word" bit 0

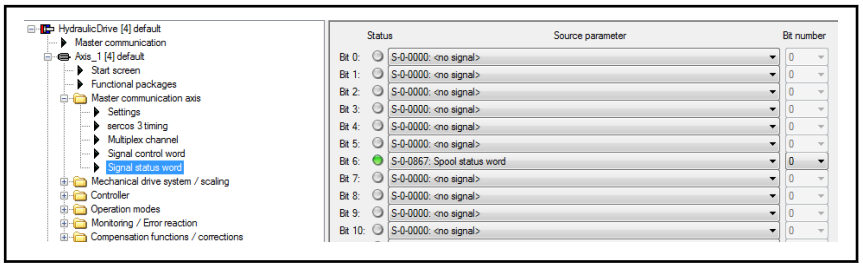


Fig. 12-51: Parameterizing the signal status word S-0-0144

- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B4	Attribute value	Attribute of the parameters S-0-0860 or P-0-2914 is not four bytes decimal with sign

Tab. 12-76: Error codes of the MB_SIIIPlcDriveDirectValveControl function block

12.5.6 MB_SIIIPlcDriveTorqueDirectValveControl

Brief description

The function block MB_SIIIPlcDriveTorqueDirectValveControl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Torque/force/direct valve control**.

Value in the operation mode parameter: 0x8051

Target system	Library
MLC/MTX for 14V20 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for HDx 20VRS or MPx 20VRS and higher	RMB_SercosIII_Util

Tab. 12-77: Reference table

Interface description

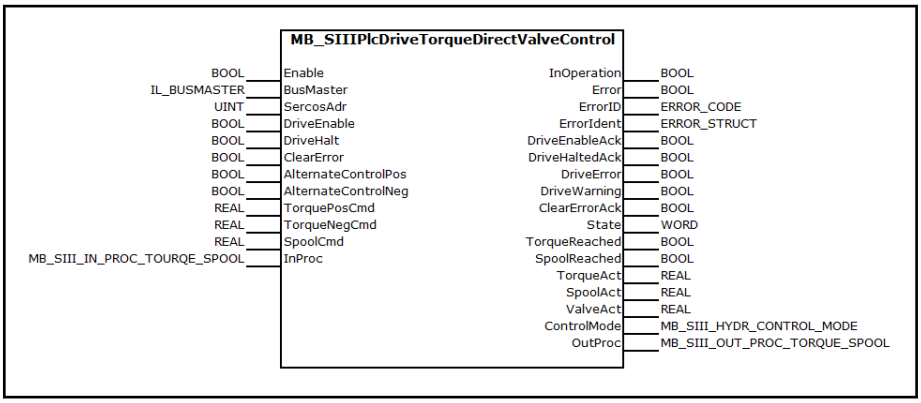


Fig. 12-52: MB_SIIIPlcDriveTorqueDirectValveControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the BusMaster input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0` Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again
	AlternateControl Pos	BOOL	Enabling alternating control in positive direction P-0-2854 " Alternating control, control word 2" bit 3 (P-0-2853 bit 0 has to be set)
	AlternateControlNeg	BOOL	Enabling alternating control in negative direction P-0-2854 " Alternating control, control word 2" bit 4 (P-0-2853 bit 0 has to be set)
	TorquePosCmd	REAL	S-0-0550, Torque/force command value, positive, scaling acc. to attribute
	Torque-NegCmd	REAL	S-0-0551, Torque/force command value, negative, scaling acc. to attribute

I/O type	Name	Data type	Description
	SpoolCmd	REAL	S-0-0860, Spool command value, scaling acc. to attribute
	InProc	MB_SIII_IN_PROC_TORQUE_SPOOL	Input structure MB_SIII_IN_PROC_TORQUE_SPOOL, page 532
VAR_OUTPUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 510
	DriveEnableAck	BOOL	Drive enabled acknowledged
	DriveHaltedAck	BOOL	Drive halt acknowledged S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Device status of IL_SIIIRemoteSlaveState
	TorqueReached	BOOL	Force command value reached (S-0-0824, Status "Torque/force command value attained" bit 0)
	SpoolReached	BOOL	Valve command value reached (S-0-0867, Spool status word bit 0)
	TorqueAct	REAL	S-0-0552, Torque/force feedback value, scaling acc. to attribute
	SpoolAct	REAL	S-0-0862, Spool feedback value, scaling acc. to attribute
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	ControlMode	MB_SIII_HYDR_CONTROL_MODE	P-0-2855 Alternating control, status word bit 0 (one force control controller active) MB_SIII_HYDR_CONTROL_MODE, page 526
	OutProc	MB_SIII_OUT_PROC_TORQUE_SPOOL	Output structure MB_SIII_OUT_PROC_TORQUE_SPOOL, page 532

Tab. 12-78: MB_SIIIPlcDriveTorqueDirectValveControl interface

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
AlternateControl Pos	BOOL			FALSE	Continuous
AlternateControlNeg	BOOL			FALSE	Continuous
TorquePosCmd	REAL	N. def.	N. def.	0.0	Continuous
TorqueNegCmd	REAL	N. def.	N. def.	0.0	Continuous
SpoolCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_TORQUE_SPOOL				Continuous

Functional description

The function block switches the drive to the operation mode "Torque/force/ direct valve control" (operation mode code: 16#8051). A valve command value is specified at the "SpoolCmd" input.

The actual valve value is shown at the output "SpoolAct".

For the alternating control, a positive torque or force command value can be specified at the "TorquePosCmd" input and a negative torque or force command value can be specified at the "TorqueNegCmd" input.

The actual torque or force value is displayed at the "TorqueAct" output.

These floating point numbers (REAL) are internally converted into fixed point values of the device (DINT) using the structures "InProc" and "OutProc".

When the valve command value is reached, this is displayed at the output "SpoolReached" (S-0-0867, Spool status word bit 0).

Reaching of the torque/force command value is displayed at the output "TorqueReached" (S-0-0824, Status "Torque/force command value attained" bit 0).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes.

The device-specific switching between the direct valve control and the torque or force control is displayed at the output "ControlMode". To do so,

P-0-2855, Alternating control, status word is evaluated.

A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word

- S-0-0144, Signal status word
- S-0-0552, Torque/force feedback value
- S-0-0862, Spool feedback value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0145, Signal control word
- S-0-0550, Torque/force command value, positive
- S-0-0551, Torque/force command value, negative
- S-0-0860, Spool command value

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

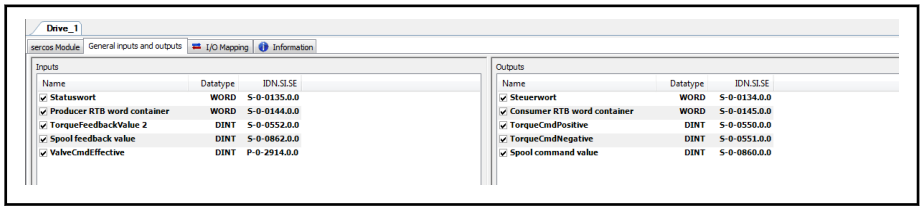


Fig. 12-53: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, this structure is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

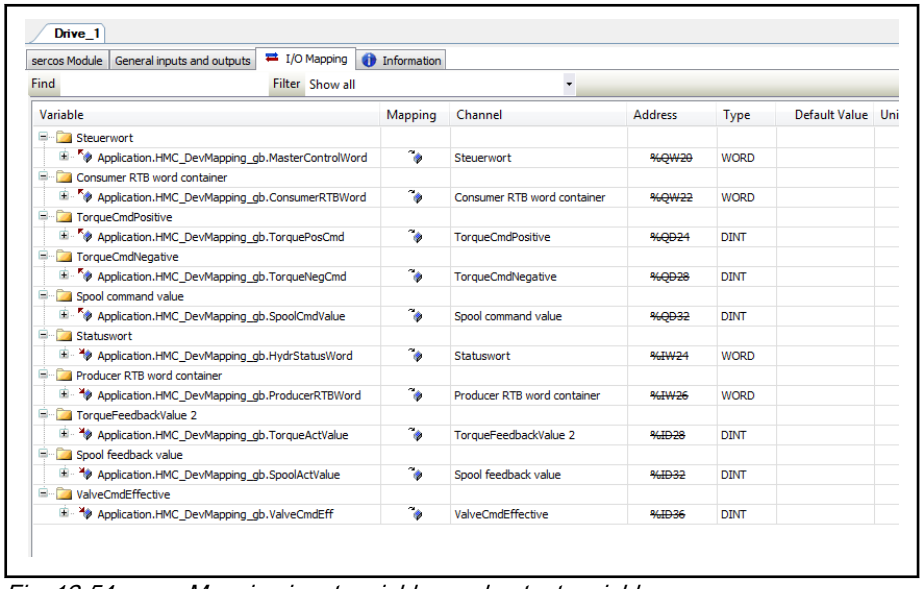


Fig. 12-54: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- Setting main operation mode to torque/force/direct valve control
  - S-0-0032 " Primary operation mode" Value "0x8051"
  - Bit 8: Set P-0-2853 Alternating control, control word 1 bit 0
- Setting parameters and bits, S-0-0144 " Signal status word"
  - …
  - Bit 6: S-0-0867 " Spool status word" bit 0
  - Bit 7: S-0-0824 " Status "Torque/force command value attained"" bit 0
  - Bit 8: P-0-2855 " Alternating control, status word" bit 0

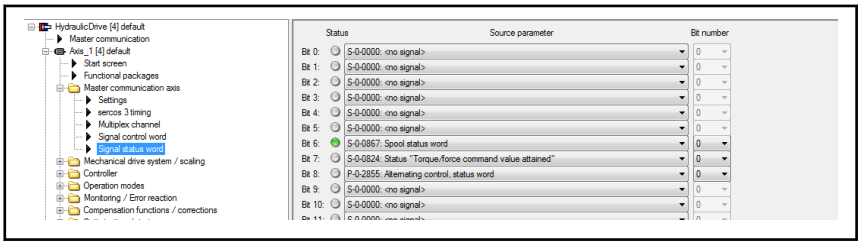


Fig. 12-55: Parameterizing the signal status word S-0-0144

- Setting parameters and bits, S-0-0145 " Signal control word"
  - Bit 0: P-0-2854 " Alternating control, control word 2" bit 3
  - Bit 1: P-0-2854 " Alternating control, control word 2" bit 4

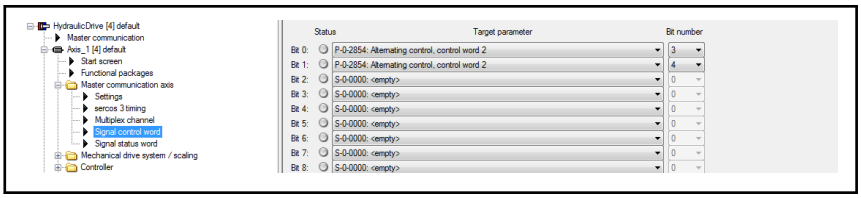


Fig. 12-56: Parameterizing the signal control word S-0-0145

- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B5	Attribute value	Attribute of the parameters S-0-0860, S-0-0550, S-0-0551 or P-0-2914 is not four bytes decimal with sign

Tab. 12-79: Error codes of the MB_SIIIPlcDriveTorqueDirectValveControl

12.5.7 MB_SIIIPlcDriveTorqueFlowControl

Brief description

The function block MB_SIIIPlcDriveTorqueFlowControl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Torque/ force/flow rate control**.

Value in the operation mode parameter: 0x8052

Target system	Library
MLC/MTX for 14V20 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for HDx 20VRS or MPx 20VRS and higher	RMB_SercosIII_Util

Tab. 12-80: Reference table

Interface description

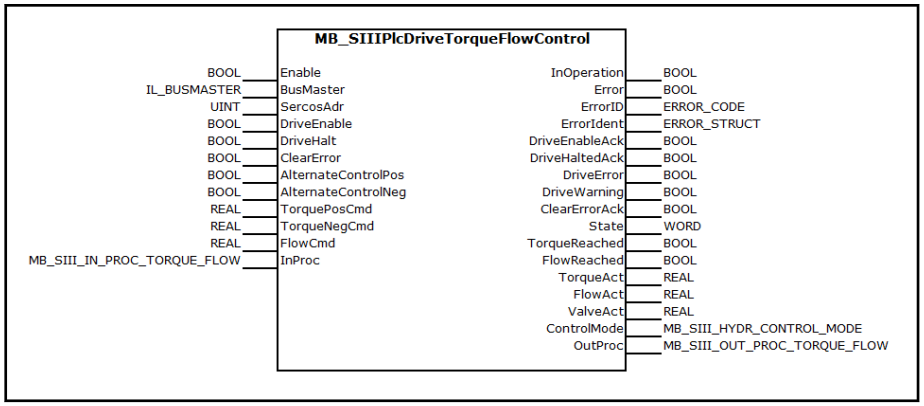


Fig. 12-57: MB_SIIIPlcDriveTorqueFlowControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the BusMaster input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0` Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13

I/O type	Name	Data type	Description
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again
	AlternateControl Pos	BOOL	Enabling alternating control in positive direction P-0-2854 " Alternating control, control word 2" bit 3 (P-0-2853 bit 0 has to be set)
	AlternateControlNeg	BOOL	Enabling alternating control in negative direction P-0-2854 " Alternating control, control word 2" bit 4 (P-0-2853 bit 0 has to be set)
	TorquePosCmd	REAL	S-0-0550, Torque/force command value, positive, scaling acc. to attribute
	TorqueNegCmd	REAL	S-0-0551, Torque/force command value, negative, scaling acc. to attribute
	FlowCmd	REAL	S-0-0840, Flow command value, scaling acc. to attribute Flow rate command value for S-0-0840
	InProc	MB_SIII_IN_PROC_TORQUE_FLOW	Input structure MB_SIII_IN_PROC_TORQUE_FLOW, page 532
VAR_OUTPUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 516
	DriveEnableAck	BOOL	Drive enabled acknowledged
	DriveHaltedAck	BOOL	Drive halt acknowledged S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Device status of IL_SIIIRemoteSlaveState
	TorqueReached	BOOL	Force command value reached (S-0-0824, Status "Torque/force command value attained" bit 0)
	FlowReached	BOOL	Flow rate command value reached (S-0-0849, Flow status word bit 1)
	TorqueAct	REAL	S-0-0552, Torque/force feedback value, scaling acc. to attribute

I/O type	Name	Data type	Description
	FlowAct	REAL	S-0-0862, Spool feedback value, scaling acc. to attribute Actual flow rate value of S-0-0862
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	ControlMode	MB_SIII_HYDR_CONTROL_MODE	P-0-2855 Alternating control, status word bit 0 (one force control controller active) MB_SIII_HYDR_CONTROL_MODE, page 526
	OutProc	MB_SIII_OUT_PROC_TORQUE_FLOW	Output structure MB_SIII_OUT_PROC_TORQUE_FLOW, page 533

Tab. 12-81: MB_SIIIPlcDriveTorqueFlowControl interface

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
AlternateControl Pos	BOOL			FALSE	Continuous
AlternateControlNeg	BOOL			FALSE	Continuous
TorquePosCmd	REAL	N. def.	N. def.	0.0	Continuous
TorqueNegCmd	REAL	N. def.	N. def.	0.0	Continuous
FlowCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_TORQUE_FLOW				Continuous

Functional description

The function block switches the drive to the operation mode "Torque/force/flow rate control" (operation mode code: 16#8052). A flow rate command value is specified at the "FlowCmd" input.

The actual value of the flow rate is displayed at the "FlowAct" output.

For the alternating control, a positive torque or force command value can be specified at the "TorquePosCmd" input and a negative torque or force command value can be specified at the "TorqueNegCmd" input.

The actual torque or force value is displayed at the "TorqueAct" output.

These floating point numbers (REAL) are internally converted into fixed point values of the device (DINT) using the structures "InProc" and "OutProc".

Reaching of the flow rate command value is displayed at the output "FlowReached" (S-0-0849, Flow status word, bit 1).

Reaching of the torque/force command value is displayed at the output "TorqueReached" (S-0-0824, Status "Torque/force command value attained" bit 0).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes.

The device-specific switching between the direct valve control and the torque or force control is displayed at the output "ControlMode". To do so,

P-0-2855, Alternating control, status word is evaluated.

A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0144, Signal status word
- S-0-0552, Torque/force feedback value
- S-0-0842, Flow feedback value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0145, Signal control word
- S-0-0550, Torque/force command value, positive
- S-0-0551, Torque/force command value, negative
- S-0-0840, Flow command value

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

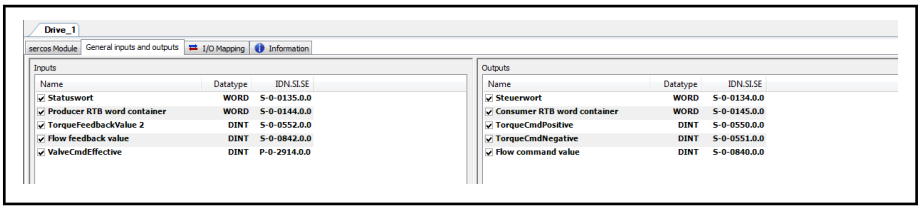


Fig. 12-58: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, it is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

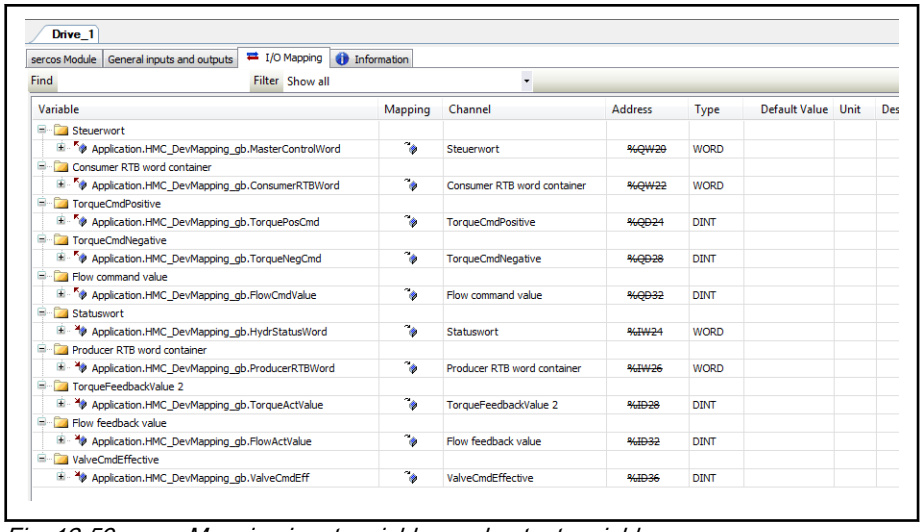


Fig. 12-59: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- Setting main operation mode to direct valve control
  - S-0-0032 " Primary operation mode" Value "0x8052"
  - Bit 8: Set P-0-2853 Alternating control, control word 1 bit 0
- Setting parameters and bits, S-0-0144 " Signal status word"
  - …
  - Bit 1 S-0-0849 " Flow status word" bit 1
  - …
  - Bit 7: S-0-0824 " Status "Torque/force command value attained"" bit 0
  - Bit 8: P-0-2855 " Alternating control, status word" bit 0

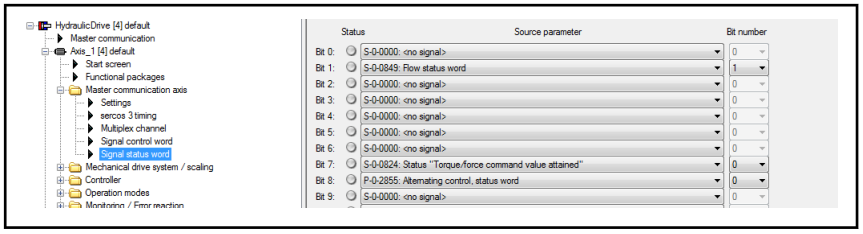


Fig. 12-60: Parameterizing the signal status word S-0-0144

- Setting parameters and bits, S-0-0145 " Signal control word"
  - Bit 0: P-0-2854 " Alternating control, control word 2" bit 3
  - Bit 1: P-0-2854 " Alternating control, control word 2" bit 4

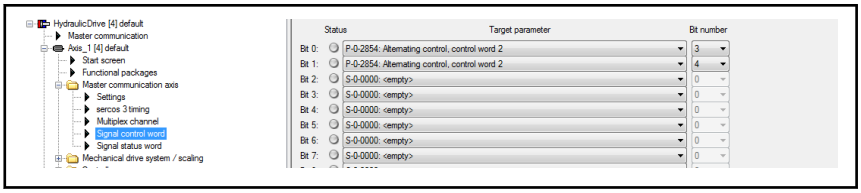


Fig. 12-61: Parameterizing the signal control word S-0-0145

- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B6	Attribute value	Attribute of the parameters S-0-0840, S-0-0550, S-0-0551 or P-0-2914 is not four bytes decimal with sign

Tab. 12-82: Error codes of the MB_SIIIPlcDriveTorqueFlowControl

12.5.8 MB_SIIIPlcDrivePressureDirectValveControl

Brief description

The function block MB_SIIIPlcDrivePressureDirectValveControl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Pressure control/Direct valve control**.

Value in the operation mode parameter: 0x8055

For the details to possible operation modes, see:

Rexroth HydraulicDrive HDx-20 Functions, Application Description, DOK-HYDRV*-HD*-20VRS**-APxx-EN-P, RD-30338-FK, chapter 6, Operation modes

Target system	Library
MLC/MTX for 14V20 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for HDx 20VRS or MPx 20VRS and higher	RMB_SercosIII_Util

Tab. 12-83: Reference table

Interface description

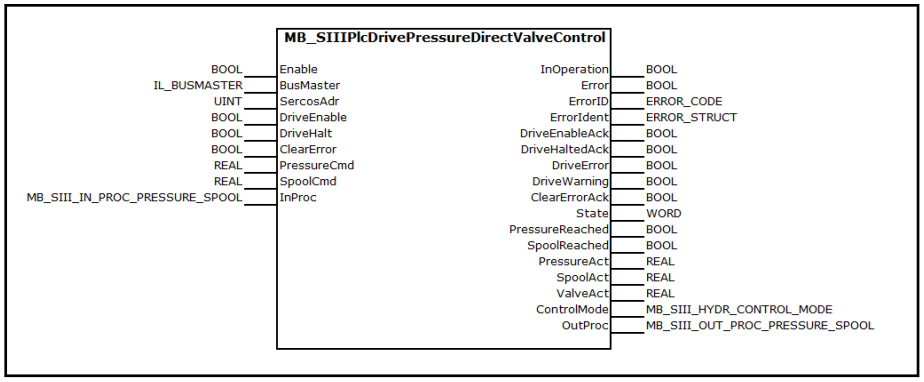


Fig. 12-62: MB_SIIIPlcDrivePressureDirectValveControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the BusMaster input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0` Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again
	PressureCmd	REAL	S-0-0800, Pressure command value, scaling acc. to attribute
	SpoolCmd	REAL	S-0-0860, Spool command value, scaling acc. to attribute
	InProc	MB_SIII_IN_PROC_PRESSURE_SPOOL	Input structure MB_SIII_IN_PROC_PRESSURE_SPOOL, page 533
VAR_OUTPUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 521
	DriveEnableAck	BOOL	Drive enabled acknowledged
	DriveHaltedAck	BOOL	Drive halt acknowledged S-0-0135, Drive status word, bit 4

I/O type	Name	Data type	Description
	DriveError	BOOL	Drive error S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Device status of IL_SIIIRemoteSlaveState
	Pressur-eReached	BOOL	Pressure command value reached (S-0-0830, Pressure status word bit 0)
	SpoolReached	BOOL	Valve command value reached (S-0-0867, Spool status word bit 0)
	PressureAct	REAL	S-0-0809, Pressure feedback value, scaling acc. to attribute
	SpoolAct	REAL	S-0-0862, Spool feedback value, scaling acc. to attribute
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	ControlMode	MB_SIII_HYDR_CON-TROL_MODE	P-0-2801, p/Q controller status word, bit 0 (1 pressure controller, 0 direct valve control active) MB_SIII_HYDR_CONTROL_MODE, page 526
	OutProc	MB_SIII_OUT_PROC_PRES-SURE_SPOOL	Output structure MB_SIII_OUT_PROC_PRESSURE_SPOOL, page 534

Tab. 12-84: MB_SIIIPlcDrivePressureDirectValveControl interface

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Bus master	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
PressureCmd	REAL	N. def.	N. def.	0.0	Continuous
SpoolCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_PRESSURE_SPOOL				Continuous

Functional description The function block switches the drive to the operation mode "Torque/direct valve control" (operation mode code: 16#8055). A valve command value is specified at the "SpoolCmd" input and a pressure command value is specified at the "PressureCmd" input.

The actual valve value is shown at the output "SpoolAct".

The actual pressure value is displayed at the output "PressureAct".

These floating point numbers (REAL) are internally converted into fixed point values of the device (DINT) using the structures "InProc" and "OutProc".

When the valve command value is reached, this is displayed at the output "SpoolReached" (S-0-0867, Spool status word bit 0).

Reaching of the pressure command value is displayed at the output "PressureReached" (S-0-0830, Pressure status word bit 0).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes.

The device-specific switching between the pressure control and the direct valve control is displayed at the output "ControlMode". To do so,

P-0-2801, p/Q controller status word is evaluated.

A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0144, Signal status word
- S-0-0809, Pressure feedback value
- S-0-0862, Spool feedback value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0800, Pressure command value
- S-0-0860, Spool command value

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

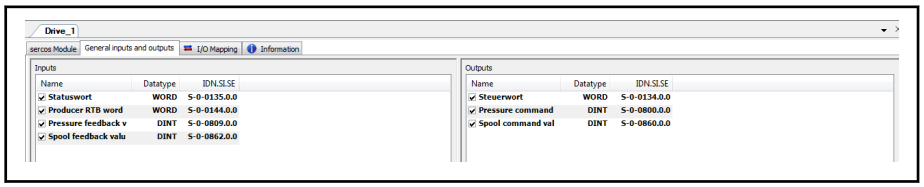


Fig. 12-63: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, this structure is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

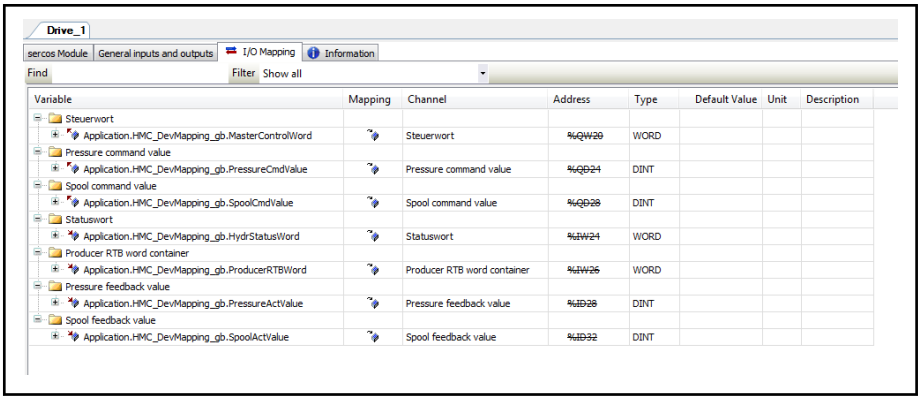


Fig. 12-64: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- Setting main operation mode to pressure control/direct valve control
  - S-0-0032 " Primary operation mode" Value "0x8055"
- Setting parameters and bits, S-0-0144 " Signal status word"
  - Bit 0: S-0-0830 " Pressure status word" bit 1
  - …
  - Bit 6: S-0-0867 " Spool status word" bit 0

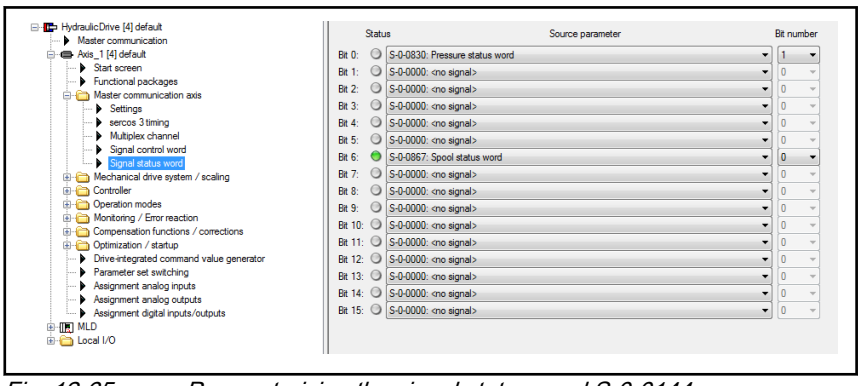


Fig. 12-65: Parameterizing the signal status word S-0-0144

- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)

Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B7	Attribute value	Attribute of the parameters S-0-0860, S-0-0800 or P-0-2914 is not four bytes decimal with sign

Tab. 12-85: Error codes of the MB_SIIIPlcDrivePressureDirectValveControl

12.5.9 MB_SIIIPlcDriveVelocityAltCtrl

Brief description

The function block MB_SIIIPlcDriveVelocityAltCtrl extends the function block MB_SIIIPlcDriveType01 and implements the operation mode **Velocity control** (with alternating force control).

Value in the operation mode parameter: 0x0002

Target system	Library
MLC/MTX for 14V20 and higher	RMB_SercosIII_Util
MLD with IndraLogic 2G for HDx 20VRS or MPx 20VRS and higher	RMB_SercosIII_Util

Tab. 12-86: Reference table

Interface description

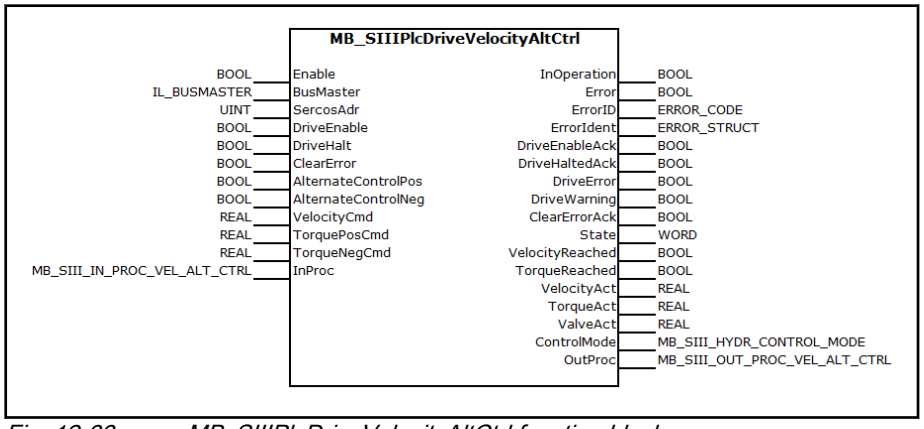


Fig. 12-66: MB_SIIIPlcDriveVelocityAltCtrl function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	Bus master	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the BusMaster input variable. If only one master is operated, it has the master `ID = IL_BUSMASTER_0` Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the slave device
	DriveEnable	BOOL	Enabling the drive (AH/AF) S-0-0134, Master control word, controller enabled, bit 14
	DriveHalt	BOOL	Stopping the drive (AH) S-0-0134, Master control word, drive halt, bit 13

I/O type	Name	Data type	Description
	ClearError	BOOL	Clearing an error in the drive by the “Clear error” command (S-0-0099, C0500 Reset class 1 diagnostics). After the error has been cleared, the selected enabling can be activated again
	AlternateControlPos	BOOL	Enabling alternating control in positive direction P-0-2854 " Alternating control, control word 2" bit 3 (P-0-2853 bit 0 has to be set)
	AlternateControlNeg	BOOL	Enabling alternating control in negative direction P-0-2854 " Alternating control, control word 2" bit 4 (P-0-2853 bit 0 has to be set)
	VelocityCmd	REAL	S-0-0036, Velocity command value
	TorquePosCmd	REAL	S-0-0550, Torque/force command value, positive, scaling acc. to attribute
	TorqueNegCmd	REAL	S-0-0551, Torque/force command value, negative, scaling acc. to attribute
	InProc	MB_SIII_IN_PROC_VEL_ALT_CTRL	Applying process input data %I MB_SIII_IN_PROC_VEL_ALT_CTRL, page 534
VAR_OUTPUT	InOperation	BOOL	Function block is operating
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 526
	DriveEnableAck	BOOL	Drive enabled acknowledged
	DriveHaltedAck	BOOL	Drive halt acknowledged S-0-0135, Drive status word, bit 4
	DriveError	BOOL	Drive error S-0-0135, Drive status word, bit 13
	DriveWarning	BOOL	Drive warning S-0-0135, Drive status word, bit 12
	ClearErrorAck	BOOL	Confirmation that the command “Clear error” was executed S-0-0099, C0500 Reset class 1 diagnostics
	State	WORD	Device status of IL_SIIIRemoteSlaveState
	VelocityReached	BOOL	Actual velocity value reached (S-0-0330)
	TorqueReached	BOOL	Force command value reached (S-0-0824, Status "Torque/force command value attained" bit 0)
	VelocityAct	REAL	S-0-0040, Velocity feedback value of encoder 1, scaling acc. to attribute
	TorqueAct	REAL	S-0-0552, Torque/force feedback value, scaling acc. to attribute

I/O type	Name	Data type	Description
	ValveAct	REAL	P-0-2914, Effective valve command value, scaling acc. to attribute
	ControlMode	MB_SIII_HYDR_CONTROL_MODE	P-0-2855 Alternating control, status word bit 0 (one force control controller active) MB_SIII_HYDR_CONTROL_MODE, page 526
	OutProc	MB_SIII_OUT_PROC_VEL_ALT_CTRL	Output structure MB_SIII_OUT_PROC_VEL_ALT_CTRL, page 535

Tab. 12-87: MB_SIIIPlcDriveVelocityAltCtrl interface

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER			FALSE	Continuous
SercosAdr	UINT	1	255	1	Rising edge at “Enable”
DriveEnable	BOOL			FALSE	Continuous
DriveHalt	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
AlternateControl Pos	BOOL			FALSE	Continuous
AlternateControlNeg	BOOL			FALSE	Continuous
VelocityCmd	REAL	N. def.	N. def.	0.0	Continuous
TorquePosCmd	REAL	N. def.	N. def.	0.0	Continuous
TorqueNegCmd	REAL	N. def.	N. def.	0.0	Continuous
InProc	MB_SIII_IN_PROC_VEL_ALT_CTRL				Continuous

Functional description

The function block switches the drive to the operation mode "Velocity control (with alternating force control)" (operation mode code: 16#0002). A velocity command value is specified at the "VelocityCmd" input.

The actual velocity value is displayed at the "VelocityAct" output.

For the alternating control, a positive torque or force command value can be specified at the "TorquePosCmd" input and a negative torque or force command value can be specified at the "TorqueNegCmd" input.

The actual torque or force value is displayed at the "TorqueAct" output.

These floating point numbers (REAL) are internally converted into fixed point values of the device (DINT) using the structures "InProc" and "OutProc".

Reaching of the velocity command value is displayed at the output "VelocityReached" (S-0-0330, Status "v_feedback = v_command", bit 0).

Reaching of the torque/force command value is displayed at the output "TorqueReached" (S-0-0824, Status "Torque/force command value attained" bit 0).

The effective valve command value (P-0-2914, Effective valve command value) is displayed at the output "ValveAct" for diagnostic purposes.

The device-specific switching between the direct valve control and the torque or force control is displayed at the output "ControlMode". To do so,

P-0-2855, Alternating control, status word is evaluated.

A (present) warning or error is additionally displayed at the function block.

Configuring cyclic data for Sercos:

To establish the cyclic communication between the function block and the drive, copy the parameters to the respective input variables and output variables. The cyclically transferred parameters are configured in the Project Explorer below the Sercos node. Therefore, select the "Drive" node below the respective devices.

These following parameters have to be configured cyclically:

Inputs:

- S-0-0135, Drive status word
- S-0-0144, Signal status word
- S-0-0040, Velocity feedback value of encoder 1
- S-0-0552, Torque/force feedback value
- P-0-2914, Effective valve command value

Outputs:

- S-0-0134, Master control word
- S-0-0145, Signal control word
- S-0-0036, Velocity command value
- S-0-0550, Torque/force command value, positive
- S-0-0551, Torque/force command value, negative

At first, configure the cyclic parameters in the "General inputs and outputs" tab.

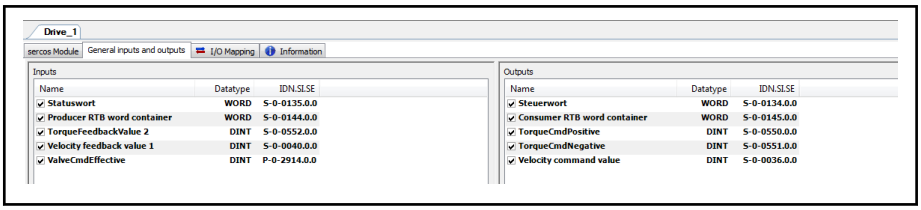


Fig. 12-67: Configuring cyclic parameters

Subsequently, the corresponding input and output variables are assigned in the "I/O mapping" tab including the instance variables of the PLC program. In the lower image, this is a global structure HPC_DevMapping_gb with the respective structural elements. Then, in the PLC program, it is copied to the structures "InProc" and "OutProc".

The structures "InProc" and "OutProc" of the function block can be entered here as well.

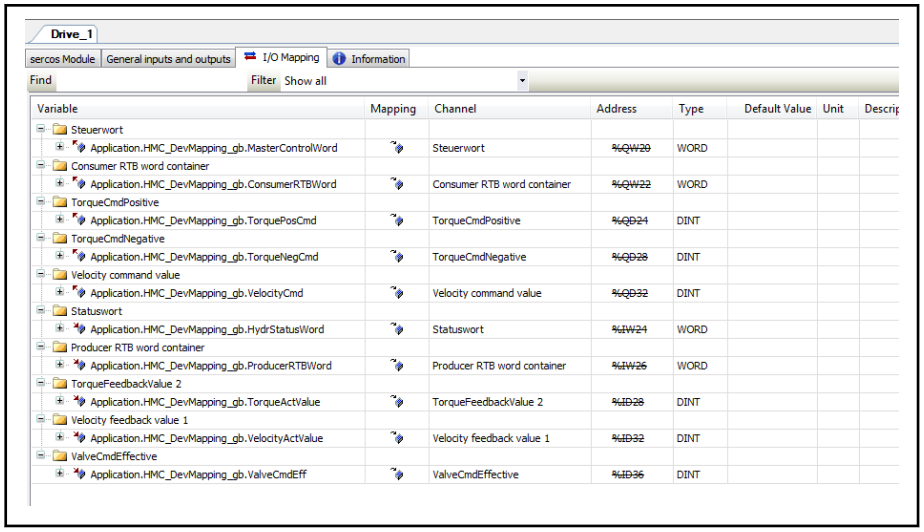


Fig. 12-68: Mapping input variables and output variables

Parameterizing the device:

For Rexroth devices, it is recommended to use "IndraWorks Ds" for the parameterization. Device-specific commissioning dialogs are implemented.

Use the entry "IndraWorks Ds here" (below the Sercos node, right-click) to start "IndraWorks Ds" in the Engineering Suite IndraWorks (automatic routing from the control to the device).

The following parameters have to be set:

- Setting main operation mode to direct valve control
  - S-0-0032 " Primary operation mode" Value "0x0002"
- Setting parameters and bits, S-0-0144 " Signal status word"
  - …
  - Bit 7: S-0-0824 " Status "Torque/force command value attained"" bit 0
  - Bit 8: P-0-2855 " Alternating control, status word" bit 0
  - Bit 9: S-0-0330 " Status "v_feedback = v_command"" bit 0

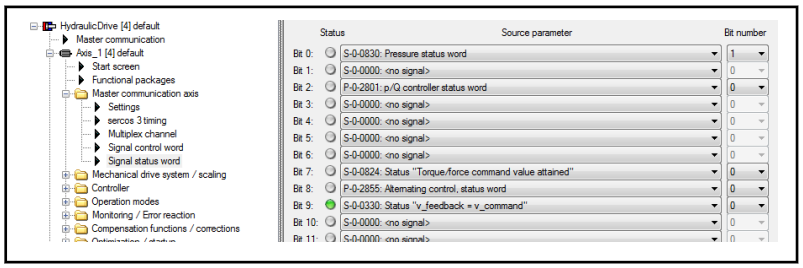


Fig. 12-69: Parameterizing the signal status word S-0-0144

- Setting parameters and bits, S-0-0145 " Signal control word"
  - Bit 0: P-0-2854 " Alternating control, control word 2" bit 3
  - Bit 1: P-0-2854 " Alternating control, control word 2" bit 4

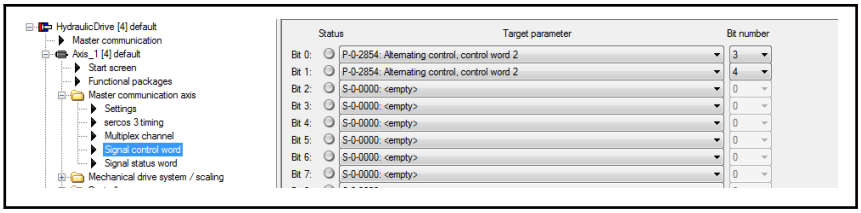


Fig. 12-70: Parameterizing the signal control word S-0-0145

- Parameterizing the device acc. to the application (measuring system, scaling, monitoring/error reaction...)



Check especially that the command values specified by the function block are not overwritten by a device-specific parameterization.

E.g. parameterizing the analog inputs of the device with internal command values.

Troubleshooting

In addition to the errors of the internally used function blocks MB_SIIIPlcDriveType01 and IL_SIIISvcRead, there are the following errors of the function block in the error table F_RELATED_TABLE (ERROR_TABLE = 16#0170).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	State	Error in state machine
INPUT_RANGE_ERROR	16#000029B7	Attribute value	Attribute of the parameters S-0-0040, S-0-0550, S-0-0551 or P-0-2914 is not four bytes decimal with sign

Tab. 12-88: Error codes of the MB_SIIIPlcDriveVelocityAltCtrl

12.5.10 MB_SIII_HYDR_CONTROL_MODE

Brief description

The enumeration type MB_SIII_HYDR_CONTROL_MODE describes the possible operation modes for Rexroth HydraulicDrives.

Name	Value	Description
NO_CONTROL_MODE	0	Default
FLOW_CONTROL_MODE	1	Closed-loop flow rate control
PRESSURE_CONTROL_MODE	2	Closed-loop pressure control
SWIVEL_ANGLE_CONTROL_MODE	3	Closed-loop swiveling angle control
VALVE_DIRECT_CONTROL_MODE	4	Direct valve control
TORQUE_CONTROL_MODE	5	Torque/force control
VELOCITY_CONTROL_MODE	6	Velocity control (with alternating force control)
POSITION_CONTROL_MODE	7	Position control

Tab. 12-89: Elements of the MB_SIII_HYDR_CONTROL_MODE enumeration type

12.5.11 MB_SIII_IN_PROC_FLOW

Brief description

The data structure MB_SIII_IN_PROC_FLOW contains the process input data of the function block MB_SIIIPlcDriveFlowControl on page 483.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word In bit 1, the parameter S-0-0849, Flow status word is configured with bit 1. (Display shows flow rate reached)
Flow	DINT	16#0	Input variable %IW of the current actual flow rate value S-0-0842, Flow feedback value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-90: Elements of the MB_SIII_IN_PROC_FLOW structure

12.5.12 MB_SIII_OUT_PROC_FLOW

Brief description The data structure MB_SIII_OUT_PROC_FLOW contains the process output data of the function block MB_SIIIPlcDriveFlowControl on page 483.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
Flow	DINT	16#0	Output variable %QW of the flow rate command value S-0-0840, Flow command value

Tab. 12-91: Elements of the MB_SIII_OUT_PROC_FLOW structure

12.5.13 MB_SIII_IN_PROC_PRESSURE_FLOW

Brief description The data structure MB_SIII_IN_PROC_PRESSURE_FLOW contains the process input data of the function block MB_SIIIPlcDrivePressureFlowControl on page 487.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word In bit 0, the parameter S-0-0830, Pressure status word is configured with bit 1. (Display shows pressure reached) In bit 1, the parameter S-0-0849, Flow status word is configured with bit 1. (Display shows flow rate reached) **Function package “Pump control”:** In bit 2, the parameter P-0-2951, Pump controller status word is configured with bit 0. (Display shows pressure or swiveling angle control active) **Function package “P/Q control”:** In bit 2, the parameter P-0-2801, p/Q controller status word is configured with bit 0. (Display shows pressure or flow rate control active)
Pressure	DINT	16#0	Input variable %IW of the current actual pressure value S-0-0809, Pressure feedback value
Flow	DINT	16#0	Input variable %IW of the current actual flow rate value S-0-0842, Flow feedback value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-92: Elements of the MB_SIII_IN_PROC_PRESSURE_FLOW structure

12.5.14 MB_SIII_OUT_PROC_PRESSURE_FLOW

Brief description The data structure MB_SIII_OUT_PROC_PRESSURE_FLOW contains the process output data of the function block MB_SIIIPlcDrivePressureFlowControl on page 487.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
Pressure	DINT	16#0	Output variable %QW of the pressure command value S-0-0800, Pressure command value
Flow	DINT	16#0	Output variable %QW of the flow rate command value S-0-0840, Flow command value

Tab. 12-93: Elements of the MB_SIII_OUT_PROC_PRESSURE_FLOW structure

12.5.15 MB_SIII_IN_PROC_SWIVEL

Brief description The data structure MB_SIII_IN_PROC_SWIVEL contains the process input data of the function block MB_SIIIPlcDrivePressureSwivelAngleControl on page 492.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word In bit 0, the parameter S-0-0830, Pressure status word is configured with bit 1. (Display shows pressure reached) In bit 3, the parameter S-0-0887, Swivel angle status word is configured with bit 1. (Display shows swiveling angle reached) In bit 4, the parameter P-0-2951, Pump controller status word is configured with bit 0. (Display shows pressure or swiveling angle control active) In bit 5, the parameter P-0-2951, Pump controller status word is configured with bit 1. (Display shows pressure or swiveling angle control active)
Pressure	DINT	16#0	Input variable %IW of the current actual pressure value S-0-0809, Pressure feedback value
SwivelAngle	DINT	16#0	Input variable %IW of the current actual swiveling angle value S-0-0882, Swivel angle actual value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-94: Elements of the MB_SIII_IN_PROC_SWIVEL structure

12.5.16 MB_SIII_OUT_PROC_SWIVEL

Brief description The data structure MB_SIII_OUT_PROC_SWIVEL contains the process output data of the function block MB_SIIIPlcDrivePressureSwivelAngleControl on page 492.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
Pressure	DINT	16#0	Output variable %QW of the pressure command value S-0-0800, Pressure command value
SwivelAngle	DINT	16#0	Output variable %QW of the swiveling angle command value S-0-0880, Swivel angle command value

Tab. 12-95: Elements of the MB_SIII_OUT_PROC_SWIVEL structure

12.5.17 MB_SIII_IN_PROC_SWIVEL_SLAVE

Brief description The data structure MB_SIII_IN_PROC_SWIVEL_SLAVE contains the process input data of the function block MB_SIIIPlcDrivePressureSwivelAngleSlave on page 497.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element. MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word In bit 3, the parameter S-0-0887, Swivel angle status word is configured with bit 1. (Display shows swiveling angle reached)
SwivelAngle	DINT	16#0	Input variable %IW of the current actual swiveling angle value S-0-0882, Swivel angle actual value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-96: Elements of the MB_SIII_IN_PROC_SWIVEL structure

12.5.18 MB_SIII_OUT_PROC_SWIVEL_SLAVE

Brief description The data structure MB_SIII_OUT_PROC_SWIVEL contains the process output data of the function block MB_SIIIPlcDrivePressureSwivelAngleSlave on page 497.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element. MB_SIII_CONTROLWORD on page 444
Pressure	DINT	16#0	Output variable %QW of the pressure command value S-0-0800, Pressure command value
SwivelAngle	DINT	16#0	Output variable %QW of the swiveling angle command value S-0-0880, Swivel angle command value

Tab. 12-97: Elements of the MB_SIII_OUT_PROC_SWIVEL_SLAVE structure

12.5.19 MB_SIII_IN_PROC_SPOOL

Brief description The data structure MB_SIII_IN_PROC_SPOOL contains the process input data of the function block MB_SIIIPlcDriveDirectValveControl on page 501.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word. S-0-0867/bit 0 (ValveComm = ValveActual) is configured
Spool	DINT	16#0	Input variable %ID of the actual valve value S-0-0862 Spool feedback value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-98: Elements of the MB_SIII_IN_PROC_SPOOL structure

12.5.20 MB_SIII_OUT_PROC_SPOOL

Brief description The data structure MB_SIII_OUT_PROC_SPOOL contains the process output data of the function block MB_SIIIPlcDriveDirectValveControl on page 501.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element MB_SIII_CONTROLWORD on page 444
Spool	DINT	16#0	Output variable %QD of the valve command value S-0-0860 Spool command value

Tab. 12-99: Elements of the MB_SIII_OUT_PROC_SPOOL structure

12.5.21 MB_SIII_IN_PROC_TORQUE_SPOOL

Brief description The data structure MB_SIII_IN_PROC_TORQUE_SPOOL contains the process input data of the function block MB_SIIIPlcDriveTorqueDirectValveControl on page 505.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word. S-0-0824 bit 0 (torque) and S-0-0867/bit 0 ($Valve_{Comm}$ = $Valve_{Actual}$) are configured
Torque	DINT	16#0	S-0-0552 Torque/force feedback value
Spool	DINT	16#0	S-0-0862 Spool feedback value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-100: Elements of the MB_SIII_IN_PROC_TORQUE_SPOOL structure

12.5.22 MB_SIII_OUT_PROC_TORQUE_SPOOL

Brief description The data structure MB_SIII_OUT_PROC_TORQUE_SPOOL contains the process output data of the function block MB_SIIIPlcDriveTorqueDirectValveControl on page 505.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element MB_SIII_CONTROLWORD on page 444
ConfControl	WORD		S-0-0145 Signal control word. Bit 3 and bit 4 are entered into P-0-2854
TorquePos	DINT		S-0-0550 Torque/force command value, positive
TorqueNeg	DINT		S-0-0551 Torque/force command value, negative
Spool	DINT		S-0-0860 Spool command value

Tab. 12-101: Elements of the MB_SIII_OUT_PROC_TORQUE_SPOOL structure

12.5.23 MB_SIII_IN_PROC_TORQUE_FLOW

Brief description The data structure MB_SIII_IN_PROC_TORQUE_FLOW contains the process input data of the function block MB_SIIIPlcDriveTorqueFlowControl on page 511.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word. S-0-0824 bit 0 (torque) and S-0-0867/bit 0 ($Valve_{Comm}$ = $Valve_{Actual}$) are configured
Torque	DINT	16#0	S-0-0552 Torque/force feedback value
Spool	DINT	16#0	S-0-0862 Spool feedback value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-102: Elements of the MB_SIII_IN_PROC_TORQUE_FLOW structure

12.5.24 MB_SIII_OUT_PROC_TORQUE_FLOW

Brief description The data structure MB_SIII_OUT_PROC_TORQUE_FLOW contains the process output data of the function block MB_SIIIPlcDriveTorqueFlowControl on page 511.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element MB_SIII_CONTROLWORD on page 444
ConfControl	WORD		S-0-0145 Signal control word. Bit 3 and bit 4 are entered into P-0-2854
TorquePos	DINT	16#0	S-0-0550 Torque/force command value, positive
TorqueNeg	DINT	16#0	S-0-0551 Torque/force command value, negative
Flow	DINT	16#0	S-0-0840 Flow command value

Tab. 12-103: Elements of the MB_SIII_OUT_PROC_TORQUE_FLOW structure

12.5.25 MB_SIII_IN_PROC_PRESSURE_SPOOL

Brief description The data structure MB_SIII_IN_PROC_PRESSURE_SPOOL contains the process input data of the function block MB_SIIIPlcDrivePressureDirectValveControl on page 516.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word. S-0-0867/bit 0 (ValveComm = ValveActual) is configured
Pressure	DINT	16#0	Input variable %ID of the actual pressure value S-0-0809 Pressure feedback value
Spool	DINT	16#0	Input variable %ID of the actual valve value S-0-0862 Spool feedback value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-104: Elements of the MB_SIII_IN_PROC_PRESSURE_SPOOL structure

12.5.26 MB_SIII_OUT_PROC_PRESSURE_SPOOL

Brief description The data structure MB_SIII_OUT_PROC_PRESSURE_SPOOL contains the process output data of the function block MB_SIIIPlcDrivePressureDirectValveControl on page 516.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element MB_SIII_CONTROLWORD on page 444
Pressure	DINT	16#0	Output variable %QD of the pressure command value S-0-0800 Pressure command value
Spool	DINT	16#0	Output variable %QD of the valve command value S-0-0860 Spool command value

Tab. 12-105: Elements of the MB_SIII_OUT_PROC_PRESSURE_SPOOL structure

12.5.27 MB_SIII_IN_PROC_VEL_ALT_CTRL

Brief description The data structure MB_SIII_IN_PROC_VEL_ALT_CTRL contains the process input data of the function block MB_SIIIPlcDriveVelocityAltCtrl on page 521.

Structural element	Type	Default value	Description
Status word	MB_SIII_STATUSWORD		Assign input variable %IW of the status word S-0-0135, Drive status word to the "wStatusword" element MB_SIII_STATUSWORD on page 445
ConfStatus	WORD		Freely configurable signal status word S-0-0144, Signal status word. S-0-0824 bit 0 (torque) and S-0-0867/bit 0 ($Valve_{Comm}$ = $Valve_{Actual}$) are configured
Velocity	DINT	16#0	S-0-0040 Velocity feedback value of encoder 1
Torque	DINT	16#0	S-0-0552 Torque/force feedback value
ValveEff	DINT	16#0	Input variable %IW of the effective valve command value P-0-2914, Effective valve command value

Tab. 12-106: Elements of the MB_SIII_IN_PROC_VEL_ALT_CTRL structure

12.5.28 MB_SIII_OUT_PROC_VEL_ALT_CTRL

Brief description The data structure MB_SIII_OUT_PROC_VEL_ALT_CTRL contains the process output data of the function block MB_SIIIPlcDriveVelocityAltCtrl on page 521.

Structural element	Type	Default value	Description
Control word	MB_SIII_CONTROLWORD		Assign output variable %QW of the control word S-0-0134, Master control word to the "wControlword" element MB_SIII_CONTROLWORD on page 444
ConfControl	WORD		S-0-0145 Signal control word. Bit 3 and bit 4 are entered into P-0-2854
TorquePos	DINT	16#0	S-0-0550 Torque/force command value, positive
TorqueNeg	DINT	16#0	S-0-0551 Torque/force command value, negative
VelocityCmd	DINT	16#0	S-0-0036 Velocity command value

Tab. 12-107: Elements of the MB_SIII_OUT_PROC_VEL_ALT_CTRL structure

12.6 RexrothSytronix

12.6.1 MB_SIIIDRn5020

Brief description The function block MB_SIIIDRn5020 allows the pressure and swivel angle control of the EFC5610 based on Syntronix DRn5020 of Rexroth controls via Sercos communication. This function block can be used to set limit values of pressure and swivel angle and the maximum flow rate, the control can be enabled/stopped and the errors of the drive can be reset. The function block includes the monitoring of different drive functions as well. These functions are described in the following sections.

Library:	RMB_SercosIII_Util
Group:	"RexrothSytronix"
Target system:	MTX 15VRS MLC 15VRS, PLC function package MLD with MPx20

Tab. 12-108: Reference table of the MB_SIIIDRn5020

Interface description

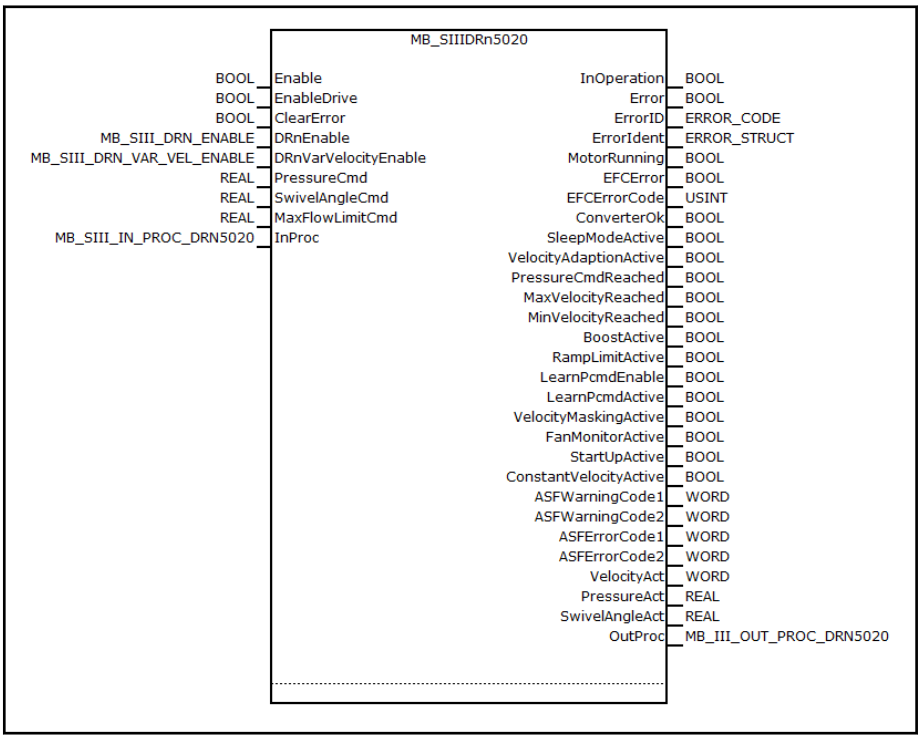


Fig. 12-71: Function block of the MB_SIIIDRn5020 interface

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled
	EnableDrive	BOOL	TRUE - Drive enabled FALSE - Disable drive
	ClearError	BOOL	Drive error, execute “Reset” command
	DRnEnable	MB_SIII_DRN_ENABLE	DRn function enabled MB_SIII_DRN_ENABLE, page 545
	DRnVarVelocityEnable	MB_SIII_DRN_VAR_VEL_ENABLE	DRn variable velocity selection MB_SIII_DRN_VAR_VEL_ENABLE, page 545
	PressureCmd	REAL	Pressure command value in bar
	SwivelAngleCmd	REAL	Swivel angle command value in %
	MaxFlowLimitCmd	REAL	Maximum flow rate limit value in rpm
	InProc	MB_SIII_IN_PROC_DRN5020	Input structure for the function block
VAR_OUTPUT	InOperation	BOOL	TRUE - Function block is operating

I/O type	Name	Data type	Description
	Error	BOOL	TRUE - Function block operation stopped due to errors
	ErrorID	ERROR_CODE	Error code for the current error
	ErrorIdent	ERROR_STRUCT	Detailed error information
	MotorRunning	BOOL	Motor status TRUE - Motor rotating FALSE - Motor is torque-free
	EFCError	BOOL	TRUE - Error active at EFC FALSE - No error
	EFCErrorCode	USINT	Error code for EFC errors
	ConverterOk	BOOL	TRUE - Converter OK FALSE - Converter not OK
	SleepModeActive	BOOL	TRUE - Sleep mode active FASLE - Inactive
	VelocityAdaptationActive	BOOL	Status of velocity adaptation TRUE - Active FASLE - Inactive
	PressureCmdReached	BOOL	TRUE - Current pressure reaches command value and is in the permitted range FALSE - Current pressure beyond permitted range
	MaxVelocityReached	BOOL	Velocity status TRUE - Maximum velocity reached FALSE - Not reached
	MinVelocityReached	BOOL	Velocity status TRUE - Minimum velocity reached FALSE - Not reached
	BoostActive	BOOL	Status of boost function TRUE - Boost active FALSE - Boost inactive
	RampLimitActive	BOOL	Status of ramp limitation TRUE - Active FASLE - Inactive
	LearnPcmdEnable	BOOL	TRUE - Learning p_cmd enabled FALSE - No enabling
	LearnPcmdActive	BOOL	TRUE - Learning p_cmd active FASLE - Inactive
	VelocityMaskingActive	BOOL	TRUE - Velocity masking active FASLE - Inactive

I/O type	Name	Data type	Description
	FanMonitorActive	BOOL	TRUE - Motor performance reduction (fan monitoring) active FASLE - Inactive
	StartUpActive	BOOL	TRUE - Commissioning active FASLE - Inactive
	ConstantVelocityActive	BOOL	TRUE - Constant speed active FASLE - Inactive
	ASFWarningCode1	WORD	Warning code 1
	ASFWarningCode2	WORD	Warning code 2
	ASFErrorCode1	WORD	Error code 1
	ASFErrorCode2	WORD	Error code 2
	VelocityAct	REAL	Current velocity in rpm
	PressureAct	REAL	Current pressure in bar
	SwivelAngleAct	REAL	Current swivel angle in %
	OutProc	MB_SIII_OUT_PROC_DRN5020	Output structure for the function block

Tab. 12-109: MB_SIIIDRn5020 interface

Min./max. and default values of the inputs

Name	Data type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
StartCommand	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
DRnEnable	MB_SIII_DRN_ENABLE			DRN_DISABLE	Continuous
DRnVarVelocityEnable	MB_SIII_DRN_VAR_VEL_ENABLE			DRN_VAR_VEL_DISABLE	Continuous
PressureCmd	REAL	0.0	1,000.0	0.0	Continuous
SwivelAngleCmd	REAL	0.00	100.00	0.00	Continuous
MaxFlowLimitCmd	REAL	0.0	10,000.0	0.0	Continuous
InProc	MB_SIII_IN_PROC_DRn5020				Continuous

Tab. 12-110: Min./max. and default values of the inputs MB_SIIIDRn5020

Function block inputs/outputs - Parameter mapping

I/O type	Name	Parameters
VAR_INPUT	EnableDrive	Parameter S-0-0134 bits 13, 14, 15
	ClearError	S-0-0099 (when using IL_SIIICommand)
	DRnEnable	Parameter F0.20 - Bit 0
	DRnVarVelocityEnable	Parameter F0.20 - Bit 1

I/O type	Name	Parameters
	PressureCmd	Parameter F0.21
	SwivelAngleCmd	Parameter F0.22
	MaxFlowLimitCmd	Parameter F0.23
VAR_OUTPUT	MotorRunning	Parameter H0.01 - Bit 1
	EFCError	Parameter H0.01 - Bit 7
	EFCErrorCode	Parameter H0.01 - Bits 8-15
	ConverterOk	Parameter H0.02 – Bit 2
	SleepModeActive	Parameter H0.02 – Bit 1
	VelocityAdaptationActive	Parameter d0.80 - Bit 1
	PressureCmdReached	Parameter d0.80 - Bit 4
	MaxVelocityReached	Parameter d0.80 - Bit 5
	MinVelocityReached	Parameter d0.80 - Bit 6
	BoostActive	Parameter d0.80 - Bit 7
	RampLimitActive	Parameter d0.80 - Bit 8
	LearnPcmdEnable	Parameter d0.80 - Bit 9
	LearnPcmdActive	Parameter d0.80 - Bit 10
	VelocityMaskingActive	Parameter d0.80 - Bit 11
	FanMonitorActive	Parameter d0.80 - Bit 12
	StartUpActive	Parameter d0.80 - Bit 13
	ConstantVelocityActive	Parameter d0.80 - Bit 14
	ASFWarningCode1	Parameter d0.86
	ASFWarningCode2	Parameter d0.87
	ASFErrorCode1	Parameter d0.88
	ASFErrorCode2	Parameter d0.89
	VelocityAct	Parameter d0.83
	PressureAct	Parameter d0.84
	SwivelAngleAct	Parameter d0.85

Tab. 12-111: Function block inputs/outputs - Parameter mapping

Functional description

The function block MB_SIIIDRn5020 uses the cyclic read/write channel to exchange data between the PLC and the drive. All required parameters are configured in the AT/MDT of the drive. Sercos is the communication protocol between the PLC and Sercos. EFC parameters have to be set for the communication. If the “Enable” input is reset at runtime, the control word data and the command values sent to the drive are not reset. They reflect their latest updated values.

This function block provides the following control functions:

- Start/stop command
- Deleting a drive error
- Activating/deactivating DRn

- Activating/deactivating the velocity adaptation
- Setting the pressure command value
- Setting the swivel angle
- Setting the maximum flow rate limit value

This function block provides the following monitoring functions:

- Motor status
- Error EFC
- Error code EFC
- Status of converter
- Status of sleep mode
- Status of velocity adaptation
- Current pressure status against command pressure
- Maximum speed reaches status
- Minimum speed reaches status
- Status of boost function
- Status of ramp limitation
- Learning p_cmd enabled
- Learning p_cmd active
- Status of velocity masking
- Status of motor performance reduction (fan monitoring)
- Status of start sequence
- Status of constant speed active
- Actual velocity
- Actual pressure
- Actual swivel angle
- ASF warning code and different warnings
- ASF error code and different errors

For details refer to Pressure control systems Sytronix DRn 5020.

Configuration of real-time input (AT) and real-time output (MDT)

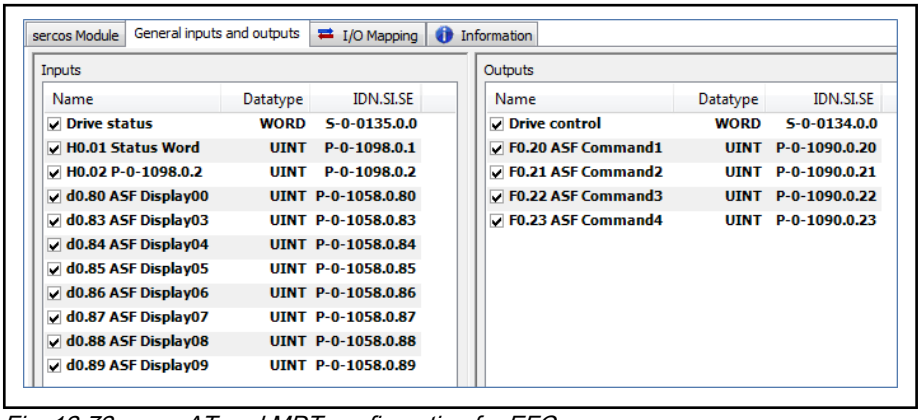


Fig. 12-72: AT and MDT configuration for EFC

Troubleshooting

The function block can go into an error state due to invalid entries. During this period, the function block reports an error and stops the operation. The control word bits and the previously set command values **are not reset** as per the

system requirements for DRn5020. The user has to correct the input values to valid values and has then to set the enable input of the function block again (default error reset procedure) to start the operation of the function block again. The error codes are listed in the following table.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#000029E0	16#00000001	DRnEnable input is invalid
INPUT_INVALID_ERROR	16#000029E1	16#00000002	DRnVarVelocityEnable input is invalid
INPUT_RANGE_ERROR	16#00000002	16#00000003	The input PressureCmd is beyond the valid range
INPUT_RANGE_ERROR	16#00000002	16#00000004	The input SwivelAngleCmd is beyond the valid range
INPUT_RANGE_ERROR	16#00000002	16#00000005	The input MaxFlowLimitCmd is beyond the valid range

Tab. 12-112: Error codes of the MB_SIIIDRn5020 function block

12.6.2 MB_SIIIFcP5020

Brief description

The function block MB_SIIIFcP5020 allows the pressure and flow control of Rexroth controls via Sercos III communication of the EFC5610 based on Syntronix FcP5020. Set the command values of pressure and flow rate using this function block. The control can thus also be enabled and stopped, the drive can be set as master or slave, the parameter group can be toggled between group 0 and group 1 for the p/Q-PID control, etc. The function block includes the monitoring of different drive functions as well. These functions are described in the following sections.

Library:	RMB_SercosIII_Util
Group:	"RexrothSytronix"
Target system:	MTX 15VRS MLC 15VRS, PLC function package MLD with MPx20

Tab. 12-113: Reference table of the MB_SIIIFcP5020

Interface description

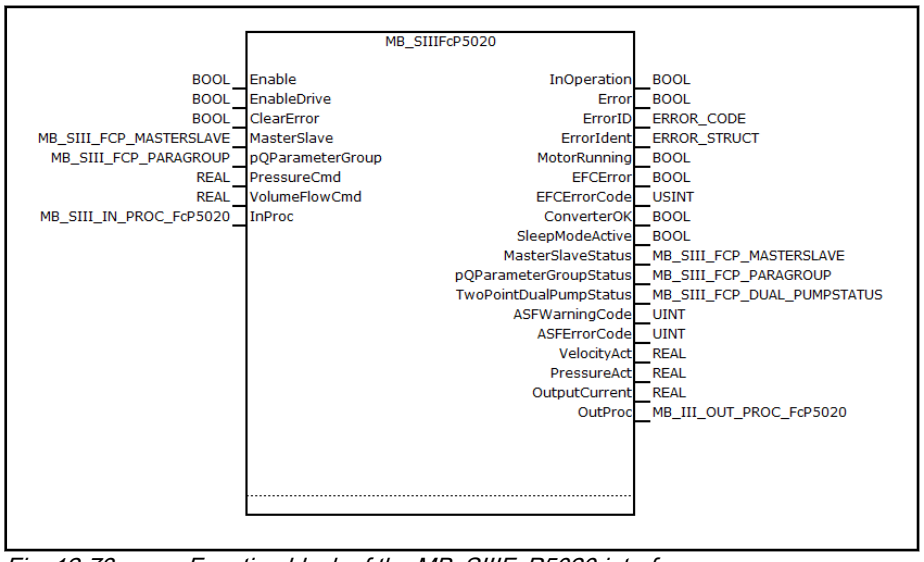


Fig. 12-73: Function block of the MB_SIIIFcP5020 interface

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled
	EnableDrive	BOOL	TRUE - Drive enabled FALSE - Disable drive
	ClearError	BOOL	Execute drive error reset command
	MasterSlave	MB_SIII_FCP_MASTER-SLAVE	Selection between master and slave operation MB_SIII_FCP_MASTERSLAVE, page 545
	pQParame-terGroup	MB_SIII_FCP_PARAGROUP	Selecting between parameter group 0 and 1 for p/Q PID control MB_SIII_FCP_PARAGROUP, page 546
	PressureCmd	REAL	Pressure command value in bar
	VolumeFlowCmd	REAL	Flow rate command value in rpm
	InProc	MB_SIII_IN_PROC_FcP5020	Input structure for the function block
VAR_OUTPUT	InOperation	BOOL	TRUE - Function block is operating
	Error	BOOL	TRUE - Function block operation stopped due to an error
	ErrorID	ERROR_CODE	Error code for the current error
	ErrorIdent	ERROR_STRUCT	Additional error information
	MotorRunning	BOOL	Motor status TRUE - Motor rotating FALSE - Motor in standstill
	EFCError	BOOL	TRUE - Error at EFC FALSE - No error
	EFCErrorCode	USINT	Error code for the current EFC error
	ConverterOK	BOOL	TRUE - Converter OK FALSE - Converter not OK
	SleepModeActive	BOOL	TRUE - Sleep mode active FASLE - Inactive
	MasterSlaveStatus	MB_SIII_FCP_MASTER-SLAVE	Master/slave status MB_SIII_FCP_MASTERSLAVE, page 545
	pQParameter-GroupStatus	MB_SIII_FCP_PARAGROUP	Parameter group for p/Q PID control MB_SIII_FCP_PARAGROUP, page 546
	TwoPointDual-PumpStatus	MB_SIII_FCP_DUALPUMP-STATUS	Status of two-point/double pump control MB_SIII_FCP_DUALPUMPSTATUS, page 546
	ASFWarningCode	UINT	ASF warning code
	ASFErrorCode	UINT	ASF error code
	VelocityAct	REAL	Current speed in rpm
	PressureAct	REAL	Current pressure in bar

I/O type	Name	Data type	Description
	OutputCurrent	REAL	Output voltage in A
	OutProc	MB_SIII_OUT_PROC_FcP5020	Output structure for the function block

Tab. 12-114: MB_SIIIFcP5020 interface

Min./max. and default values of the inputs

Name	Data type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
StartCommand	BOOL			FALSE	Continuous
ClearError	BOOL			FALSE	Continuous
MasterSlave	MB_SIII_FCP_MASTERSLAVE			FCP_MASTER	Continuous
pQParameterGroup	MB_SIII_FCP_PARAGROUP			FCP_PARAGROUP	Continuous
PressureCmd	REAL	0.0	1,000.0	0.0	Continuous
VolumeFlowCmd	REAL	0	5,000	0	Continuous
InProc	MB_SIII_IN_PROC_FcP5020				Continuous

Tab. 12-115: Min./max. and default values of the inputs MB_SIIIFcP5020

Function block inputs/outputs - Parameter mapping

I/O type	Name	Parameters
VAR_INPUT	EnableDrive	Parameter S-0-0134 bits 13, 14, 15
	ClearError	S-0-0099 (when using IL_SIIICommand)
	MasterSlave	Parameter F0.20 - Bit 2
	pQParameterGroup	Parameter F0.20 - Bit 3
	PressureCmd	Parameter F0.21
	VolumeFlowCmd	Parameter F0.22
VAR_OUTPUT	MotorRunning	Parameter H0.01 - Bit 1
	EFCError	Parameter H0.01 - Bit 7
	EFCErrorCode	Parameter H0.01 - Bits 8-15
	ConverterOk	Parameter H0.02 – Bit 2
	SleepModeActive	Parameter H0.02 – Bit 1
	MasterSlaveStatus	Parameter d0.80 ASF status word bit 2
	pQParameterGroupStatus	Parameter d0.80 ASF status word bit 3
	TwoPointDualPumpStatus	Parameter d0.80 ASF status word bit 10
	ASFWarningCode	Parameter d0.88

I/O type	Name	Parameters
	ASFErrorCode	Parameter d0.89
	VelocityAct	Parameter d0.01
	PressureAct	Parameter d0.82
	OutputCurrent	Parameter d0.11

Tab. 12-116: Function block inputs/outputs - Parameter mapping

Functional description

The function block MB_SIIIFcP5020 uses the cyclic read/write channel to exchange data between the PLC and the drive. All required parameters are configured in the AT/MDT of the drive. Sercos is the communication protocol between the PLC and Sercos. EFC parameters have to be set for the communication. If the “Enable” input is reset at runtime, the command values sent to the drive are not reset. They reflect their latest updated values.

This function block provides the following control functions:

- Start/stop command
- Resetting drive error
- Selecting between master/slave operation
- Selecting between parameter group 0 and 1 for p/Q PID control
- Setting pressure command value
- Setting flow rate command value

This function block provides the following monitoring functions:

- Motor status
- Error EFC
- Error code EFC
- Converter status
- Status of sleep mode
- Master/slave status
- Active parameter group for p/Q PID control
- Status of two-point/double pump control
- ASF warning code
- ASF error code
- Actual speed
- Actual pressure
- Output voltage

For details refer to Pressure control systems Sytronix FcP 5020.

Configuration of real-time input (AT) and real-time output (MDT)

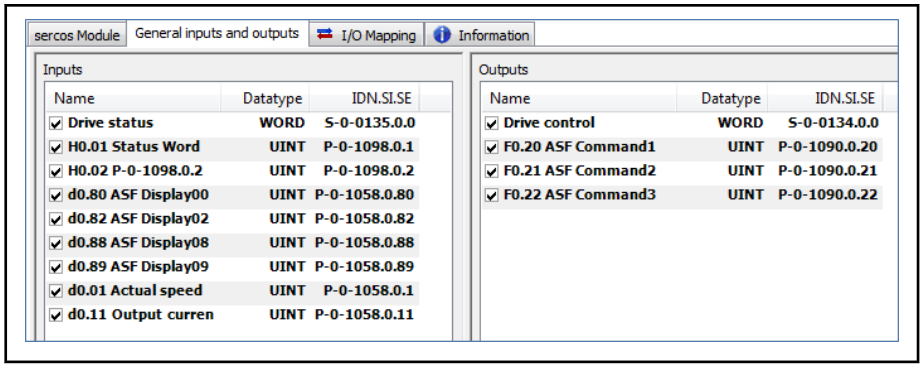


Fig. 12-74: AT and MDT configuration for EFC

Troubleshooting

The function block can go into an error state due to invalid entries. During this period, the function block reports an error and stops the operation. The control word bits and the previously set command values **are not reset** as per the system requirements for FcP5020. The user has to correct the input values to valid values and has then to set the enable input of the function block again (default error reset procedure) to start the operation of the function block again. The error codes are listed in the following table.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#000029E2	16#00000001	MasterSlave input invalid
INPUT_INVALID_ERROR	16#000029E3	16#00000002	pQParameterGroup input invalid
INPUT_RANGE_ERROR	16#00000002	16#00000003	The input PressureCmd is beyond the valid range
INPUT_RANGE_ERROR	16#00000002	16#00000004	The input VolumeFlowCmd is beyond the valid range

Tab. 12-117: Error codes of the MB_SIIIFcP5020 function block

12.6.3 MB_SIII_DRN_ENABLE

This enumeration is used by the DRnEnable input.

Element	Description
DRN_DISABLE	DRn enabling (X3) - Inactive
DRN_ENABLE	DRn enabling (X3) - Active

Tab. 12-118: Structural description of MB_SIII_DRN_ENABLE

12.6.4 MB_SIII_DRN_VAR_VEL_ENABLE

This enumeration is used by the DRnVarVelocityEnable input.

Element	Description
DRN_VAR_VEL_DISABLE	DRn variable velocity enabling (X4) - Inactive
DRN_VAR_VEL_ENABLE	DRn variable velocity enabling (X4) - Active

Tab. 12-119: Structural description of MB_SIII_DRN_VAR_VEL_ENABLE

12.6.5 MB_SIII_FCP_MASTERSLAVE

This enumeration is used by the MasterSlave input.

Element	Description
FCP_MASTER	Master
FCP_SLAVE	Slave

Tab. 12-120: Structural description of MB_SIII_FCP_MASTERSLAVE

12.6.6 MB_SIII_FCP_PARAGROUP

This enumeration is used by the pQParameterGroup input and by the pQParameterGroupStatus output.

Element	Description
FCP_PARA_GROUP0	Parameter group 0
FCP_PARA_GROUP1	Parameter group 1

Tab. 12-121: Structural description of MB_SIII_FCP_PARAGROUP

12.6.7 MB_SIII_FCP_DUALPUMPSTATUS

This enumeration is used by the TwoPointDualPumpStatus output.

Element	Description
FCP_DUALPUMP_VGMAX	Vg_{max}
FCP_DUALPUMP_VGMIN	Vg_{min}

Tab. 12-122: Structural description of MB_SIII_FCP_DUALPUMPSTATUS

13 RIL_SercosIIIDeviceReplacement.library

13.1 Introduction and overview

The RIL_SercosIIIDeviceReplacement.library provides function blocks for a device replacement without PC. It is for example possible to determine the current device configuration and to compare it to the previously saved configuration. If there are changes, the parameterization and the firmware of the devices can be restored.

Target systems

The library can be used in the following systems:

Target system	Library
MLC	RIL_SercosIIIDeviceReplacement

Overview

The functionality for the Sercos device replacement is used to replace defective devices in the Sercos bus without using a PC. The configuration of the system is backed up and then used as future reference. This reference is compared to the current configuration for future restarts or upon request. If there is a deviation, the configuration of the reference is restored. The reference of the configuration is also called "finger print", as this reference is used to clearly identify the system. The functionality for the Sercos device replacement is not a single function block that can simply be used and thus not suitable for all applications. It is a collection of function blocks to be interconnected to obtain a matching functionality for the application.

There are three different functionality groups:

- Function blocks to read and write the finger print, to read the current configuration and to compare the finger print to the current configuration
- Saving and restoring firmware and parameter data as well as rebooting devices
- Auxiliary functions, e.g. logging

Use cases

The are multiple use cases for the device replacement functionality:

- The failed device is to be replaced. Subsequently, the defined firmware and parameters have to be written to the device
- Multiple failed devices have to be replaced
- All devices have to be replaced

  This is only a virtual case in a system, but it can be used for the serial commissioning. In this case, a system would be set up with all devices. Subsequently, all files of an existing system are copied to the new system and the device replacement functionality has to be started. This functionality can detect that all devices have changed and it will update the devices accordingly.

Distinction - Unsupported cases

The term “Device replacement” is present for multiple functionalities. The previous section thus only described the supported use cases. For a better understanding, cases, which are not supported explicitly, are now listed. These are:

- Restoring safety-related parameters. These are included in a parameter backup for Bosch Rexroth drives and are thus also loaded into the devices during the restoration, but more actions are required to enable safety-related parameters. This is not part of the device replacement
- Changed topology:

The device replacement functionality is based on the concept that the devices are located at the topologically correct address. If the topology is changed, the device replacement functionality assumes that the devices changed and tries to restore them. The complete topology is for example inverted when interchanging the plugs at the two Sercos ports of the control. The consequence is that the device replacement functionality tries to restore all devices.

- Compatible hardware:

  It is possible that newer compatible hardware is used during a device replacement. It can replace the old one, but the device replacement functionality detects the new hardware as being different and will thus abort with an error. In this case, the device has to be replaced manually. Subsequently, a new reference configuration has to be created.

Reference configuration/finger print and current configuration

The reference configuration or the finger print as well as the current configuration are significant for the functionality. The finger print has to be saved remanently and is thus stored as file on the CF card. Path and name of the file can be specified. The file includes actual device data, control data and type data. For troubleshooting und checking purposes, the file is stored as text file in .csv format.

⚠ WARNING **Loss of data**

The .csv format is completely text-based format that separates values using commas. Microsoft Excel also uses this format. Thus, it can also be opened with Microsoft Excel. Microsoft Excel converts some data. Thus, data can become incorrect. If this data is saved, loss of data or the loss of the device replacement functionality can result. It is thus recommended to open the file with a simple text editor.

The current configuration does not have to be saved, as the configuration can read out directly. However, this is useful for comparisons to save the configuration remanently. Path names and file names can be specified.

Database

A "database" is used for the finger print and for the current configuration. This database is an internal data management allowing to add or remove devices and to perform different sortings. These are background tasks. However, it is required to open the program "IL_DB_ControlAsync" from the directory "400_Database/PRG" in a task of low priority. It is recommended to create that task as free-running type.

No further action is required for the operation. Only the path and name of the database have to be specified. For function blocks that have to access the database, a variable of the type IL_DB_TYPE has to be created additionally used to manage the database.

Time response

The device replacement function is a functionality that should not be used for a system at runtime, but only when starting or upon request. Data is also stored as file, parameters are written and read and the firmware is replaced if required. These actions might require a lot of time. Thus, the device replacement functionality should be in a task of low priority. It can also be task in which the program for the database is opened.

⚠ CAUTION **Automatic switching of the Sercos phase**

For most actions, the device replacement functionality has to be in the Sercos phases 0 or 2. Thus, the function blocks change the phase automatically. Additionally, the devices are rebooted when exchanging the firmware and are thus not available at the Sercos bus for a short period. Thus, it is required before the start to ensure that the system is in a state which can use the device replacement function without causing damages (e.g. all axes are in standstill).

Example of an automated device replacement

There are different use cases for the device replacement functionality. An example is included in the library. The code in the "_Example" directory is located in the library manager and can be copied for other purposes. It also includes a small visualization that can be used to visualize the individual steps again. This example is described in detail in the next section.

Preparations

Some prerequisites have to be met to use the device replacement functionality.

Opening the programs

A database is used in the device replacement functionality to manage device data. It is required to start a background program for the database. The background program is stored in the library in the directory "400_Database/PRG". The program is "IL_DB_ControlAsync". It is recommended to create that task as free-running type of low priority.

It is also required to implement the device replacement functionality in a task. It is recommended to use another program. This program can also be opened in the same task used for the background program of the database.

Dynamic memory

To execute the device replacement functionality, dynamic memory is required. It is used for the dynamic creation of function blocks. To create the dynamic memory, go to the context menu of the “Application” node and select the properties. A dialog opens. Go to the "Application build options" dialog. The dynamic memory allocation can be enabled and the maximum memory size can be specified:

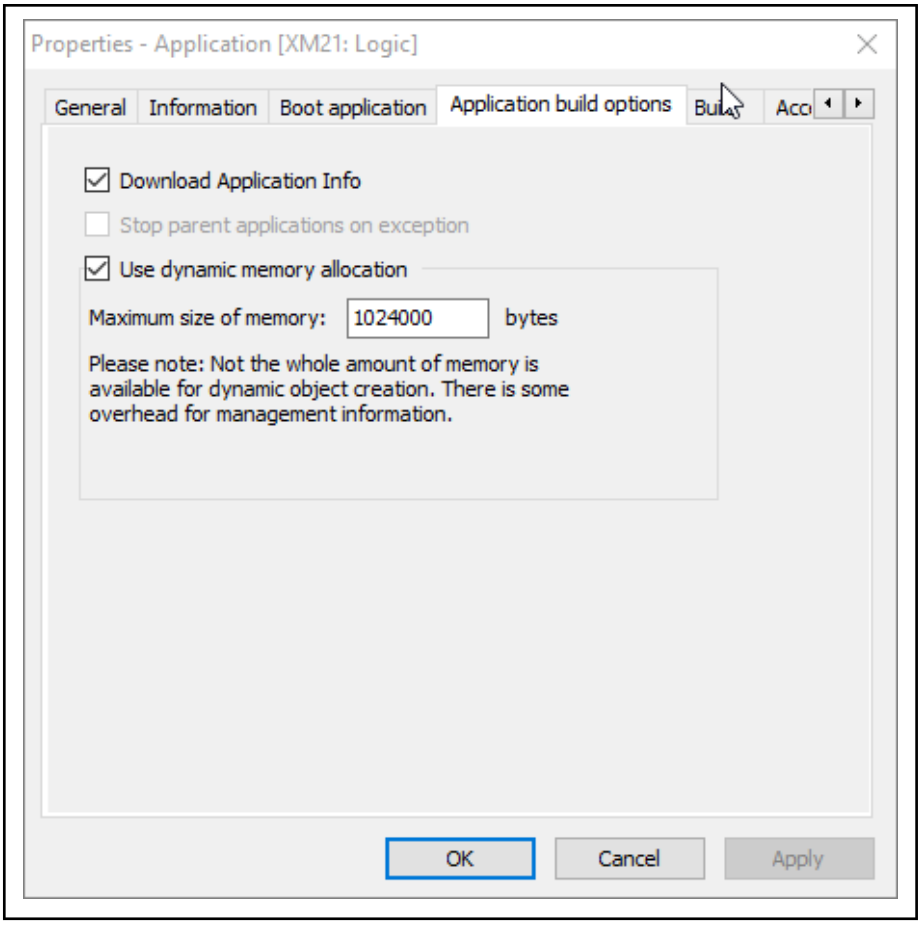


Fig. 13-1: Dialog to enter the size of the dynamic memory

The size of the dynamic memory depends on the number of the devices to be replaced and the how fast they should be replaced. At least 130,000 bytes should be provided. Only one device can be edited (the parameter “Backup/restore” or the firmware “Backup/restore”). If there are 1,000,000 bytes, up to nine devices can be edited parallely. The more memory is available, the more devices can be edited parallely and the faster the processing. The maximum number is 32 devices.

If a defective device is to be replaced, it is recommended to provide 130,000 bytes.

If multiple devices used are replaced simultaneously, a memory of 1,000,000 is reasonable with regard to memory consumption and parallel processing.

Example program declaration

The example program is not included in the library. It can be installed as GAT plug-in. The program is analyzed here to better understand the function.

Variable declaration

The program inputs are used to control its function:

- "Check":

  If the variable is set, the reference configuration is loaded, then the current configuration created and subsequently, both configurations are compared. The result of the comparison can be read at the outputs NofSercosAddrDifferent, NofParametersDifferent and NofFirmwareDifferent.

- "Init":

If the variable is set, the reference and the current configuration are loaded and if there are differences, it is tried to restore the reference configuration. The Sercos addresses are restored at the bus and if required, the firmware and parameter files are loaded to the devices.

- "Store":

  If the variable is set, the current configuration is saved as new reference configuration.

The constant "CHECK_FIRMWARE" can control whether the firmware is also to be backed up and loaded if required (= TRUE) or if not required. The parameters are always backed up and loaded.

Example program - Implementation

At the beginning of the program, the Boolean variables are initialized first for the display in the visualization and the “Start” state of the state machine is set with regard to the selected function. The state machine starts. In the STATE_BOOTING state, it is waited that the control is not in its booting state anymore. Subsequently, it is switched to the Sercos phase 2 in STATE_SWITCH_P2 state. It is ensured that there is no motion. Then, the device replacement functionality is started in the STATE_LOAD_REF state. The reference configuration is read from the CF card. The result is stored in the SercosRefList.

Actually, the current configuration is read now. If a device has already been replaced, it is very probable that the Sercos address of this device is set incorrectly. Thus, it is first tried to set all Sercos addresses in the STATE_RESTORE_BUS state using the reference configuration. Only little time is required. The current bus configuration is then read in the STATE_CREATE_ACT state. The configuration includes the corrected Sercos addresses.

In the STATE_GET_ACT state, the currently read configuration is then loaded into an array of structures. This action is not required for the device replacement, as only data from the database "SercosActList" is copied to an array of structures. Thus, it is also possible that the user uses the data. The device replacement functionality uses only the database instead.

In the next state (STATE_COMPARE), the reference and the current configuration are compared. A list is returned. All information for the device replacement is included. The differences are collected in the index IL_SDR_CONST.ALL (= -1) and specified for all devices. Thus, only this index has to be checked for a first comparison. Index 0 includes the differences for the control. The indices 1-x include the differences of each device in topological order. The function block to compare the configurations is also provided with the binary output "Difference". There are differences if this output is set. If this output is not set, there are no differences and no further actions are required.

The meaning of the individual structural elements in the list are listed in the following:

Content and meaning of the comparative list

Structural element	Parameters used	Note
RestoreHW	S-0-1300.0.1, S-0-1300.0.4, S-0-1300.0.5, S-0-1026, S-0-1300.0.11, S-0-1300.0.3	It is a linking of multiple parameters that do not change if the hardware is the same. All parameters have to be identical. If the value 65535 (= 16#FFFF) is entered, the number of the devices at the Sercos ring differ. The bus configuration cannot be restored
RestoreFW	S-0-1300.0.9 and TFTP=TRUE	This is a linking from firmware string and the TFTP support, as the TFTP is required for the firmware update.
RestorePAR	It is a combination of the changed S-0-0531 and RestoreFW	The parameter “Restore” is required if the parameter checksum changes or after a firmware update
ComponentName	S-0-1300.0.1	
VendorCode	S-0-1300.0.3	
DevDesc	S-0-1300.0.4	
DeviceId	S-0-1300.0.5	
HW_String	S-0-1300.0.8	
FW_String	S-0-1300.0.9	
OrderInfo	S-0-1300.0.11	
ManufDate	S-0-1300.0.13	
HW_Ident	S-0-1026	
FW_String	S-0-1300.0.9 alternatively S-0-0030	The parameter S-0-1300.0.9 is read. If the reading fails, S-0-0030 is read instead
TFTP	S-0-1000.0.0 includes 16#0E01=TFTP	Value contains either TRUE (if 16#0E01 is found in the parameter list) or FALSE (if not found)
ParChkSum	S-0-0531	

It is then changed to the STATE_DONE state if there are no differences. If there are differences, it is either changed to the STATE_RESTORE_FW state (changed firmware) or to the STATE_RESTORE_PAR state (changed parameters). It is always rebooted (STATE_REBOOT state) and the parameters are reloaded after a firmware change. The device replacement is now completed. The function blocks used in the “Restore” states require information to update the devices accordingly. The already known list of differences created during the comparison is used. The user has to transfer only the list with the differences. For an individual comparison (e.g. use of a different firmware starting from a specific serial number), the user has to update the firmware/parameters.

When updating the firmware and the parameters, errors can result for the individual devices. Lists are transferred to the function blocks. In case of

error, this function block fills the lists. The Sercos address and all resulting errors are included.

There are three more states (STATE_CREATE_REF, STATE_BACKUP_FW, STATE_BACKUP_PAR) in the state machine used when saving a new reference configuration. The names are specified accordingly. First, the reference configuration is created and stored as file, then, the firmware and the parameters of all devices are backed up.

Errors in the example code are not analyzed in detail. It is only jumped to the respective error state. Only status messages are generated and the program outputs are set. Further actions can take place in a real application.

For more details, refer to the description file of the GAT plugin. If there are further optimizations in the code of the GAT plugin, these are also described.

Individual comparison

In the example code, devices are replaced due to defined comparisons. In case of a changed firmware version, the firmware is changed for example. For a customized comparison, this action and the subsequent individual action (firmware/parameter update) has to be executed by the user.

Comparison

The function block to compare the configurations returns a list. This list is not analyzed in the index IL_SDR_CONST.ALL (= -1), but has to go from index 1 up to MB_MAX_SERCOS_ADDRESS (or the total length of the list). It can then be checked and analyzed individually for each device (each index) whether an action should follow. In each index, only a 1 (difference) or a 0 (no difference) exists for all elements.

If the firmware or parameter should be updated when there is a change, the update has to be made individually as well. The function blocks IL_SDR_RestoreFwGroup or IL_SDR_RestoreParGroup are used in the example code. Use these function blocks only if the complete list of the comparison is used. For an update of individual devices, use the function blocks IL_SDR_RestoreFwDevice or IL_SDR_RestoreParDevice. Transfer the structure IL_SDR_DEV_INFO_TYPE. The user has to fill this structure.

Content of the library RIL_SercosIIIDeviceReplacement

The library RIL_SercosIIIDeviceReplacement includes multiple function blocks. These are divided into sections using a folder structure. This document does not describe the function blocks in detail. The documentation is already provided in the library. For the documentation, go to the library manager.

RIL_SercosIIIDeviceReplacement function blocks

Area of application (folder in library)	
Function blocks/data types	**Description**
IL_SDR_CONFIG	Configuration data for the device replacement functionality. The user can modify path and file names. Change the other values only if requested by a Bosch Rexroth employee
Data types (100_Type)	
IL_SDR_DEVICE_REF	Structure to uniquely address a Sercos device
IL_SDR_DEV_DIFF_TYPE	Structure to list the differences between two configurations
IL_SDR_DEV_INFO_TYPE	Data structure describing a Sercos device. This data structure can also be saved and compared to other configurations
IL_SDR_MSG_LOG_TYPE	Structure to define “Logging” messages

Global Variables (200_Var)	
IL_SDR_DATA	Global data messages logger and used dynamic memory
IL_SDR_ERR_CODE	Global constant data to define error codes
Administration function blocks (300_POU/310_Administration)	
IL_SDR_CompareCofig	Function block to compare two configurations
IL_SDR_CreateDeviceList	Function block to create the current configuration at the Sercos bus
IL_SDR_CreateReferenceList	Function block to create a reference configuration
IL_SDR_CreateUserList	Function block to overwrite a configuration with modified data
IL_SDR_GetDeviceList	Reads a configuration and returns it as an array of structures. This allows a simple device data access
IL_SDR_OpenReferenceList	Function block to open a reference configuration from the memory medium
Backup/restore function blocks for individual devices (300_POU/320_Single device)	
IL_SDR_BackupFwDevice	Backing up a firmware of a device on the control
IL_SDR_BackupParDevice	Backing up a parameter set of a device on the control
IL_SDR_RestoreFwDevice	Restoring a firmware of a device from the control
IL_SDR_RestoreParDevice	Restoring a parameter set of a device from the control
Backup/restore function blocks for groups of devices (300_POU/330_ Device Group)	
IL_SDR_BackupFwGroup	Backing up firmwares of a group of devices on the control
IL_SDR_BackupParGroup	Backing up parameter sets of devices on the control
IL_SDR_RebootGroup	Reboots all devices of a group
IL_SDR_RestoreFwGroup	Restoring the firmware of a group of devices from the control
IL_SDR_RestoreParGroup	Restoring parameter sets of devices from the control
Target-dependent function blocks (300_POU/380_ TargetSpecific)	
IL_SDR_ResetBusConfig_MLC	Writes the Sercos address to all devices at the Sercos bus (address specification). Only for the MLC control!
IL_SDR_ResetBusConfig_MTX	Writes the Sercos address to all devices at the Sercos bus (address specification). Only for the MTX control (not yet implemented)!
Sercos phase (300_POU/390_ Tools/391_SIII phase)	
IL_SDR_GetPhase	Provides the current Sercos phase
IL_SDR_SetPhase	Sets the Sercos phase
Sercos phase (300_POU/390_ Tools/394_Logger)	
IL_SDR_ErrorText	Converting error numbers into text
IL_SDR_MessageLog	Function to record text messages
Database (400_Database)	
IL_DB_CONFIG	Configuration data for the database functionality. Change the values only if requested by a Bosch Rexroth employee

Database (400_Database/PRG)	
IL_DB_ControlAsync	Program to process database queries. Has to be called in a low-priority task
Database data types (400_Database/TYPE)	
IL_DB_DESCRIPTION_TYPE	Data type to describe database entries
IL_DB_TYPE	Data type to describe a database
Database “Global data” (400_Database/VAR)	
IL_DB_ERR_CODE	Definitions of error numbers

Tab. 13-1: RIL_SercosIIIDeviceReplacement library

14 RIL_InlineIOModules.library

14.1 Introduction and overview

This library describes the access options of the IEC program on Inline I/O modules.

Implementation of the library is based on the library RIL_Utilities.

The basic function blocks (basic classes) IL_EnableDoneBase, IL_EnableInOpBase, IL_ExecuteDoneBase and IL_ExecuteInOpBase are used and their methods are extended or overwritten.

Especially for complex terminals, support by coordinated function blocks for IndraLogic (2G) is important.

Use this function blocks to operate the terminals in only a few steps.

The library can be used for the following I/O modules:

Analog input terminals	Part number	Function block
R-IB IL AI 2/SF-PAC Rexroth Inline Terminal with Two Analog Input Channels	Data Sheet R911170606 Terminal R911170784	IL_AI2Sf on page 560
R-IB IL AI 2/SF-230-PAC Rexroth Inline Terminal with Two Analog Input Channels	Data Sheet R911170522 Terminal R911170425	IL_AI2Sf on page 560
R-IB IL AI 4/EF-PAC Rexroth Inline Terminal with Four Analog Differential Input Channels	Data Sheet R911170542 Terminal R911170426	IL_AI4Ef on page 564
R-IB IL AI 4/I-PAC Rexroth Inline Terminal with Four Inputs to Connect Current Signals	Data Sheet R911336655 Terminal R911172894	IL_AI4IU on page 569
R-IB IL AI 4/U-PAC Rexroth Inline terminals with four inputs to connect voltage signals	Data Sheet R911336657 Terminal R911172895	IL_AI4IU on page 569
R-IB IL AI 8/SF-PAC Rexroth Inline Terminal with Eight Analog Input Channels	Data Sheet R911170484 Terminal R911308493	IL_AI8SfIs on page 573

Analog input terminals	Part number	Function block
R-IB IL AI 8/IS-PAC Rexroth Inline Terminal with Eight Analog Input Channels	Data Sheet R911170486 Terminal R911308494	IL_AI8SfIs on page 573
R-IB IL SGI 2/F-PAC Rexroth Inline Terminal with Two Analog Inputs for Strain Gauges	Data Sheet R911170582 Terminal R911170432	IL_Sgi2f on page 578 IL_Sgi2fOffset on page 584
R-IB IL TEMP 2 RTD-PAC Rexroth Inline Terminal with Two Inputs for Temperature Sensors	Data Sheet R911170604 Terminal R911170785	IL_Temp2Rtd on page 589
R-IB IL TEMP 2 UTH-PAC Rexroth Inline Terminal with Two Inputs for Thermocouples	Data Sheet R911170518 Terminal R911170431	IL_Temp2Uth on page 584
R-IB IL TEMP 4/8 RTD-PAC Rexroth Inline Terminal with Eight Resistive Temperature Sensor Inputs	Data Sheet R911170514 Terminal R911170428	IL_Temp4_8Rtd on page 595
R-IB IL TEMP 4/8 RTD-EF-PAC Rexroth Inline terminal with eight resistive temperature sensor inputs, supports additional 4-wire sensors compared to the R-IB IL TEMP 4/8 RTD-PAC	Data Sheet R911338008 Terminal R911173029	IL_Temp4_8RtdEf on page 601

Tab. 14-1: Analog input terminals and their function blocks

Analog output terminals	Part number	Function block
R-IB IL AO 1/SF-PAC Rexroth Inline Terminal with One Analog Output	Data Sheet R911170602 Terminal R911170787	IL_AO1Sf on page 607
R-IB IL AO 2/SF-PAC Rexroth Inline Terminal with Two Analog Outputs	Data Sheet R911170520 Terminal R911170436	IL_AO2SfBp on page 610
R-IB IL AO 2/U/BP-PAC	Data Sheet Terminal R911170786	IL_AO2SfBp on page 610

Analog output terminals	Part number	Function block
R-IB IL AO 2/UI-PAC Rexroth Inline Terminal with Two Analog Outputs	Data Sheet R911338449 Terminal R911173634	IL_AO2UI on page 614

Tab. 14-2: Analog output terminals and their function blocks

Function terminals	Part number	Function block
R-IB IL CNT-PAC Rexroth Inline Power Terminal	Data Sheet R911170546 Terminal R911170788	IL_CntCount on page 618 IL_CntFrequency on page 624 IL_CntPulse on page 630 IL_CntTime on page 634
R-IB IL PWM/2-PAC Rexroth Inline Terminal for Pulse Width and Frequency Modulation IL PWM 2/CC-3A	Data Sheet R911170576 Terminal R911170444	IL_Pwm2Frequency on page 639 IL_Pwm2Pwm on page 642 IL_Pwm2PulseDirection on page 646 IL_Pwm2SinglePulse on page 649
R-IB IL SSI-PAC Rexroth Positioning Terminal for Absolute Encoders	Data Sheet R911318641 Terminal R911308594	IL_Ssi on page 652
R-IB IL SSI-IN-PAC Rexroth Inline Terminal for Absolute Encoders with SSI Interfaces	Data Sheet R911170584 Terminal R911171514	IL_SsiIn on page 667
R-IB IL INC-IN-PAC Rexroth Inline Terminal for Incremental Encoders	Data Sheet R911170488 Terminal R911308491	IL_IncIn on page 672

Tab. 14-3: Function modules and their function blocks

Communication modules		Function block
R-IB IL RS232-PRO-PAC Rexroth Inline Terminal for Serial Data Transmission	Data Sheet R911170480 Terminal R911170440	IL_RsCom on page 677
R-IB IL RS485/422-PRO-PAC Rexroth Inline Terminal for Serial Data Transmission	Data Sheet R911170482 Terminal R911170442	IL_RsCom on page 677

Communication modules		Function block
R-IB IL RS UNI-PAC Rexroth Inline Terminal for Serial Data Transmission One serial input and output channel in RS-485/422 or RS-232 design	Data Sheet R911338451 Terminal R911173635	IL_RsCom on page 677

Tab. 14-4: Communication modules and their function blocks

INLINEIO_TABLE 16#0190 The function blocks use the error table INLINEIO_TABLE, 16#0190.

14.2 Analog input terminals

14.2.1 IL_AI2Sf

Brief description The **IL_AI2Sf** function block configures the Inline terminal **R-IB IL AI 2/SF** with two analog input channels and normalizes the data read by the terminal with regard to the user-defined limit values.

One function block instance is required for each Inline I/O terminal.

Interface description

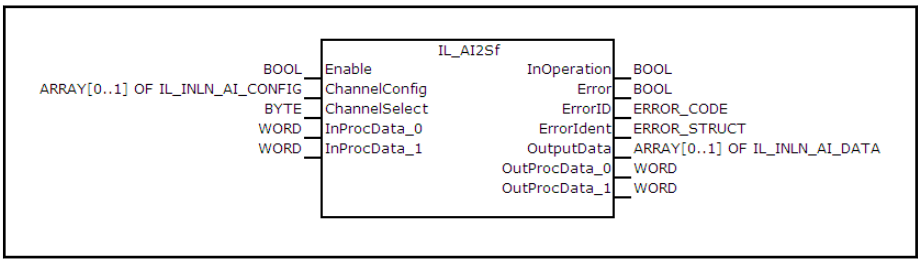


Fig. 14-1: IL_AI2Sf function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	ARRAY[0..1] OF IL_INLN_AI_CONFIG	Contains configuration data for each individual channel, see "IL_INLN_AI_CONFIG" on page 561
	ChannelSelect	BYTE	Channel selection, BYTE#2#000000yx; x - Channel 1 y - Channel 2
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 564
	OutputData	ARRAY[0..1] OF IL_INLN_AI_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_AI_DATA" on page 562
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-5: IL_AI2Sf interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	0	3	3	Positive edge at "Enable"

Tab. 14-6: Input behavior of the IL_AI2Sf

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_AI_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values of channel 1 and 2 of the terminal, set **ChannelSelect** accordingly.

Both channels are selected by default. The channels can be read with the enabled bits of **ChannelSelect**. The bits 0 and 1 correspond to the channels 1 und 2.

The measured values are normalized according to the user-defined range limits and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

IL_INLN_AI_CONFIG

The structure contains information on one channel configuration. The channel is defined by **HighLimit**, **LowLimit**, **NoOfSamples** and **InputRange**.

It is configured as follows:

```
ParameterName [Channel Number].[Function].
```

e.g. to configure **InputRange** when calling channel 1 for the first time:

```
ChannelConfig [0].InputRange.
```

Name	Type	Description	Min. value	Max. value	Default value	Effective
HighLimit	REAL	Upper limit of the normalized value	-65535.0	65535.0	10.0	Continuously with "Enable"
LowLimit	REAL	Lower limit of the normalized value	-65535.0	65535.0	0.0	Continuously with "Enable"
NoOfSamples	BYTE	0 – No average value generation 16 – Average value generation using 16 measured values	0	16	16	Positive edge at "Enable"
InputRange	IL_INLN_AI_RANGE	Measuring range, see "IL_INLN_AI_RANGE" on page 562	UNIPOLAR10V	UNIPOLAR4TO20MA	UNIPOLAR10V	Positive edge at "Enable"

Tab. 14-7: Permitted values of IL_INLN_AI_CONFIG for IL_AI2Sf

IL_INLN_AI_DATA

The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value with regard to upper and lower limit value
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 562

Tab. 14-8: Structural elements IL_INLN_AI_DATA

IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-9: Enum IL_INLN_DIAG_CODE

IL_INLN_AI_RANGE

The enumeration contains the measuring ranges to configure one channel.

ENUM	Description
UNIPOLAR10V	Voltage measurement 0 ... 10 V
BIPOLAR10V	Voltage measurement -10 V ... +10 V
UNIPOLAR20MA	Current measurement 0 ... 20 mA
BIPOLAR20MA	Current measurement -20 mA .... +20 mA
UNIPOLAR4TO20MA	Current measurement 4 ... 20 mA

Tab. 14-10: Permitted values of IL_INLN_AI_RANGE for IL_AI2Sf

Example In the following example, channel 1 is configured for a range between 0 V and 10 V and channel 2 for a range between 4 mA and 20mA.

Declaration of the example

```
PROGRAM PrgIL_AI2Sf
VAR
    fbIL_AI2Sf                : IL_AI2Sf;
    rNormValCh1                : REAL;
    rNormValCh2                : REAL;
    enDiagCodeCh1             : IL_INLN_DIAG_CODE;
    enDiagCodeCh2             : IL_INLN_DIAG_CODE;
    wInProcessData_0    : WORD;
    wInProcessData_1    : WORD;
    wOutProcessData_0    : WORD;
    wOutProcessData_1    : WORD;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_AI2Sf.ChannelConfig[0].HighLimit := REAL#10.0 ;
 fbIL_AI2Sf.ChannelConfig[0].LowLimit := REAL#0.0 ;
 fbIL_AI2Sf.ChannelConfig[0].InputRange := IL_INLN_AI_RANGE.UNIPOLAR10V ;
 fbIL_AI2Sf.ChannelConfig[0].NoOfSamples := BYTE#16 ;
 fbIL_AI2Sf.ChannelConfig[1].HighLimit := REAL#20.0 ;
 fbIL_AI2Sf.ChannelConfig[1].LowLimit := REAL#4.0 ;
 fbIL_AI2Sf.ChannelConfig[1].InputRange := IL_INLN_AI_RANGE.UNIPOLAR4TO20MA ;
 fbIL_AI2Sf.ChannelConfig[1].NoOfSamples := BYTE#16 ;
 fbIL_AI2Sf.InProcData_0 := wInProcessData_0 ;
 fbIL_AI2Sf.InProcData_1 := wInProcessData_1 ;
 wOutProcessData_0 := fbIL_AI2Sf.OutProcData_0 ;
 wOutProcessData_1 := fbIL_AI2Sf.OutProcData_1 ;
 fbIL_AI2Sf.ChannelSelect := BYTE#2#00000011;


 // FB call
 fbIL_AI2Sf(Enable:=TRUE);


 // Read data
IF fbIL_AI2Sf.InOperation = TRUE THEN
 rNormValCh1 := fbIL_AI2Sf.OutputData[0].NormalizedValue ;
 rNormValCh2 := fbIL_AI2Sf.OutputData[1].NormalizedValue ;
 enDiagCodeCh1 := fbIL_AI2Sf.OutputData[0].DiagCode ;
 enDiagCodeCh2 := fbIL_AI2Sf.OutputData[1].DiagCode ;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_AI2Sf".

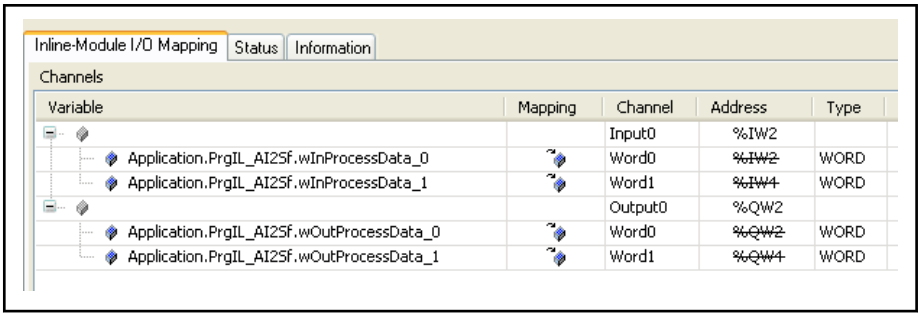


Fig. 14-2: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003004	Channel number	The "InputRange" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003005	Channel number	The "NoOfSamples" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003009	Channel number	Invalid range limits HighLimit <= LowLimit.
DEVICE_ERROR (16#0008)	16#00003020	16#000x8010	Invalid configuration, Additional2 ‘x’ shows the channel number

Tab. 14-11: Error codes of the IL_AI2Sf

14.2.2 IL_AI4Ef

Brief description

The **IL_AI4Ef** function block configures the Inline terminal **R-IB IL AI 4/EF** with four analog input channels and normalizes the data read by the terminal with regard to the user-defined limit values.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline and Profibus.

Interface description

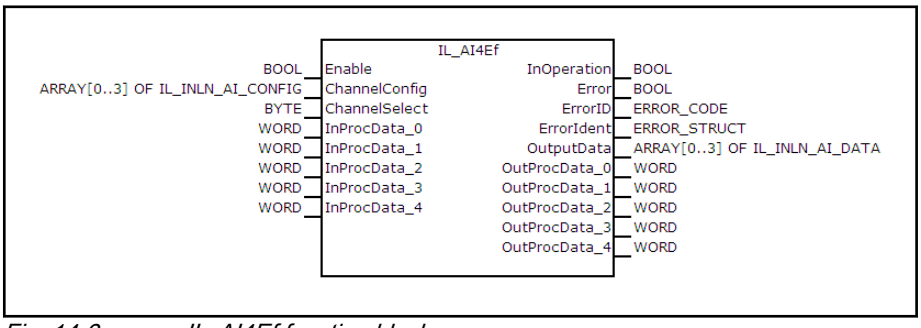


Fig. 14-3: IL_AI4Ef function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	ARRAY[0..3] OF IL_INLN_AI_CONFIG	Contains configuration data for each individual channel, see "IL_INLN_AI_CONFIG" on page 566
	ChannelSelect	BYTE	Channel selection, Byte#2#00001111; Automatic channel reading is individually activated (bit 0... bit 3 correspond to channel 1...4)
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
	InProcData_2	WORD	Process input word 2
	InProcData_3	WORD	Process input word 3
	InProcData_4	WORD	Process input word 4
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 568
	OutputData	ARRAY[0..3] OF IL_INLN_AI_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_AI_DATA" on page 566
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1
	OutProcData_2	WORD	Process output word 2
	OutProcData_3	WORD	Process output word 3
	OutProcData_4	WORD	Process output word 4

Tab. 14-12: IL_AI4Ef interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	0	15	15	Positive edge at "Enable"

Tab. 14-13: Input behavior of the IL_AI4Ef

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_AI_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values or diagnostic information of the channels 1 ... 4 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and up to bit 3 to channel 4.

All channels are selected by default.

The measured values are normalized according to the user-defined range limits and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

IL_INLN_AI_CONFIG The structure contains information on one channel configuration. The channel is defined by **HighLimit**, **LowLimit**, **NoOfSamples** and **InputRange**.

It is configured as follows:

```
ParameterName[Channel_Number].[Function].
```

e.g. to configure **InputRange** when calling channel 1 for the first time:

```
ChannelConfig[0].InputRange.
```

Name	Type	Description	Min. value	Max. value	Default value	Effective
HighLimit	REAL	Upper limit of the normalized value	-65535.0	65535.0	10.0	Continuously with "Enable"
LowLimit	REAL	Lower limit of the normalized value	-65535.0	65535.0	0.0	Continuously with "Enable"
NoOfSamples	BYTE	0 – No average value generation 4 – Average value generation using 4 measured values 16 – Average value generation using 16 measured values 32 – Average value generation using 32 measured values	0	32	16	Positive edge at "Enable"
InputRange	IL_INLN_AI_RANGE	Measuring range, see "IL_INLN_AI_RANGE" on page 567	UNIPOLAR10V	UNIPOLAR4TO20MA	UNIPOLAR10V	Positive edge at "Enable"

Tab. 14-14: Permitted values of IL_INLN_AI_CONFIG for IL_AI4Ef

IL_INLN_AI_DATA The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

```
ParameterName[Channel_Number].[Function]
```

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value between upper and lower limit value
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 567

Tab. 14-15: Structural elements IL_INLN_AI_DATA

IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-16: Enum IL_INLN_DIAG_CODE

IL_INLN_AI_RANGE

The enumeration contains the measuring ranges to configure one channel.

ENUM	Description
UNIPOLAR10V	Voltage measurement 0 ... 10 V
BIPOLAR10V	Voltage measurement -10 V ... +10 V
UNIPOLAR5V	Voltage measurement 0 … 5 V
BIPOLAR5V	Voltage measurement -5 V ... +5 V
UNIPOLAR20MA	Current measurement 0 ... 20 mA
BIPOLAR20MA	Current measurement -20 mA .... +20 mA
UNIPOLAR4TO20MA	Current measurement 4 ... 20 mA

Tab. 14-17: Permitted values of IL_INLN_AI_RANGE for IL_AI4Ef

Example

In the following example, channel 1 is configured for the range +/- 10 V and channel 2 for a range between 0 mA and 20 mA. Data is located between the upper and lower limit and available as normalized variable values.

Declaration of the example

```
PROGRAM PrgIL_AI4Ef
VAR
    fbIL_AI4Ef: IL_AI4Ef;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wInProcessData_2: WORD;
    wInProcessData_3: WORD;
    wInProcessData_4: WORD;
    wOutProcessData_0: WORD;
```

```
    wOutProcessData_1: WORD;
    wOutProcessData_2: WORD;
    wOutProcessData_3: WORD;
    wOutProcessData_4: WORD;
    rNormalizedValue_0: REAL;
    rNormalizedValue_1: REAL;
    enDiagCode_0: IL_INLN_DIAG_CODE;
    enDiagCode_1: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration of the channels 1 and 2
fbIL_AI4Ef.ChannelConfig[0].InputRange := IL_INLN_AI_RANGE.BIPOLAR10V;
fbIL_AI4Ef.ChannelConfig[0].NoOfSamples := BYTE#32;
fbIL_AI4Ef.ChannelConfig[0].HighLimit := REAL#10.0;
fbIL_AI4Ef.ChannelConfig[0].LowLimit := REAL#-10.0;
fbIL_AI4Ef.ChannelConfig[1].InputRange := IL_INLN_AI_RANGE.UNIPOLAR20MA;
fbIL_AI4Ef.ChannelConfig[1].NoOfSamples := BYTE#16;
fbIL_AI4Ef.ChannelConfig[1].HighLimit := REAL#20.0;
fbIL_AI4Ef.ChannelConfig[1].LowLimit := REAL#0.0;
fbIL_AI4Ef.ChannelSelect := BYTE#2#00000011;
fbIL_AI4Ef.InProcData_0 := wInProcessData_0;
fbIL_AI4Ef.InProcData_1 := wInProcessData_1;
fbIL_AI4Ef.InProcData_2 := wInProcessData_2;
fbIL_AI4Ef.InProcData_3 := wInProcessData_3;
fbIL_AI4Ef.InProcData_4 := wInProcessData_4;
wOutProcessData_0 := fbIL_AI4Ef.OutProcData_0;
wOutProcessData_1 := fbIL_AI4Ef.OutProcData_1;
wOutProcessData_2 := fbIL_AI4Ef.OutProcData_2;
wOutProcessData_3 := fbIL_AI4Ef.OutProcData_3;
wOutProcessData_4 := fbIL_AI4Ef.OutProcData_4;

 // FB call
fbIL_AI4Ef(Enable:= TRUE);

 // Read data
IF fbIL_AI4Ef.InOperation = TRUE THEN
 rNormalizedValue_0 := fbIL_AI4Ef.OutputData[0].NormalizedValue;
 rNormalizedValue_1 := fbIL_AI4Ef.OutputData[1].NormalizedValue;
 enDiagCode_0 := fbIL_AI4Ef.OutputData[0].DiagCode;
 enDiagCode_1 := fbIL_AI4Ef.OutputData[1].DiagCode;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_AI4Ef".

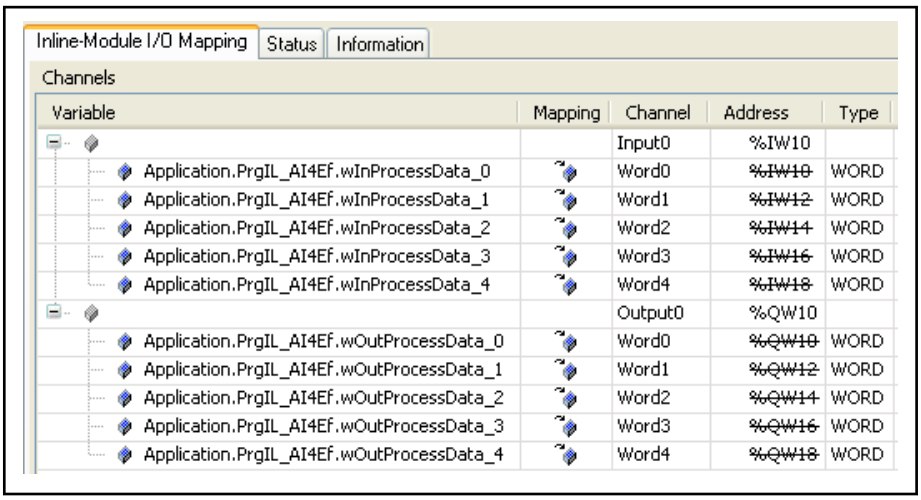


Fig. 14-4: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003004	Channel number	The "InputRange" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003005	Channel number	The "NoOfSamples" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003009	Channel number	Invalid range limits HighLimit <= LowLimit
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#00003020	16#00008010	Invalid configuration

Tab. 14-18: Error codes of the IL_AI4Ef

14.2.3 IL_AI4IU

Brief description

The **IL_AI4IU** function block configures the Inline terminals **R-IB IL AI 4/EF** and **R-IB IL AI 4/U** with four analog input channels and normalizes the data read by the terminals with regard to the user-defined limit values.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus and Profinet IO field bus.

Interface description

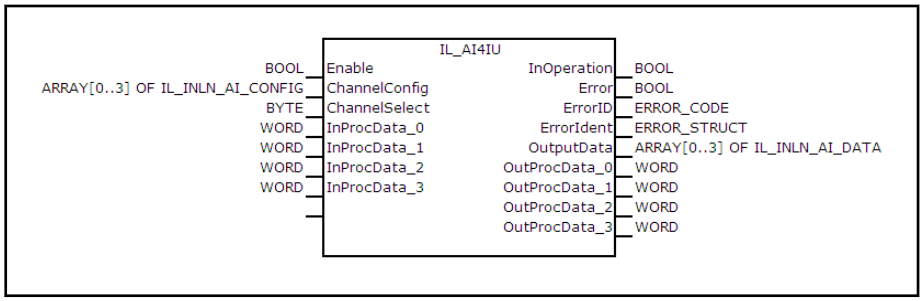


Fig. 14-5: IL_AI4IU function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	ARRAY[0..3] OF IL_INLN_AI_CONFIG	Contains configuration data for each individual channel, see "IL_INLN_AI_CONFIG" on page 570
	ChannelSelect	BYTE	Byte#2#0000xxxx; Automatic channel reading is individually activated (bit 0... bit 3 correspond to channel 1...4)
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
	InProcData_2	WORD	Process input word 2
	InProcData_3	WORD	Process input word 3

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 573
	OutputData	ARRAY[0..3] OF IL_INLN_AI_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_AI_DATA" on page 571
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1
	OutProcData_2	WORD	Process output word 2
	OutProcData_3	WORD	Process output word 3

Tab. 14-19: IL_AI4IU interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	1	15	15	Positive edge at "Enable"

Tab. 14-20: Input behavior of the IL_AI4IU

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_AI_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values or diagnostic information of the channels 1 ... 4 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and up to bit 3 to channel 4.

All channels are selected by default.

The measured values are normalized according to the user-defined range limits and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

IL_INLN_AI_CONFIG

The structure contains information on one channel configuration. The channel is defined by **HighLimit**, **LowLimit**, **NoOfSamples** and **InputRange**.

It is configured as follows:

`ParameterName[Channel_Number].[Function].`

e.g. to configure **InputRange** when calling channel 1 for the first time:

`ChannelConfig[0].InputRange`.

Name	Type	Description	Min. value	Max. value	Default value	Effective
HighLimit	REAL	Upper limit of the normalized value	-65535.0	65535.0	10.0	Continuously with "Enable"
LowLimit	REAL	Lower limit of the normalized value	-65535.0	65535.0	0.0	Continuously with "Enable"
NoOfSamples	BYTE	0 – No average value generation 4 – Average value generation using 4 measured values 16 – Average value generation using 16 measured values 32 – Average value generation using 32 measured values	0	32	16	Positive edge at "Enable"
InputRange	IL_INLN_AI_RANGE	Measuring range, see "IL_INLN_AI_RANGE" on page 572	UNIPOLAR10V	UNIPOLAR4TO20MA	UNIPOLAR10V	Positive edge at "Enable"

Tab. 14-21: Permitted values of IL_INLN_AI_CONFIG for IL_AI4IU

IL_INLN_AI_DATA

The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value between upper and lower limit value
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 571

Tab. 14-22: Structural elements IL_INLN_AI_DATA

IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration

DiagCode	ENUM	Description
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-23: Enum IL_INLN_DIAG_CODE

IL_INLN_AI_RANGE The enumeration contains the measuring ranges to configure one channel.

ENUM	Description
UNIPOLAR10V	Voltage measurement 0 ... 10 V
BIPOLAR10V	Voltage measurement -10 V ... +10 V
UNIPOLAR20MA	Current measurement 0 ... 20 mA
UNIPOLAR4TO20MA	Current measurement 4 ... 20 mA

Tab. 14-24: Permitted values of IL_INLN_AI_RANGE for IL_AI4IU

Example In the following example, channel 1 is configured for a range between 0 V and 10 V and channel 2 for a range +/- 10V. Data is located between the upper and lower limit and available as normalized variable values.

Declaration of the example

```
PROGRAM PrgIL_AI4IU
VAR
    fbIL_AI4IU: IL_AI4IU;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wInProcessData_2: WORD;
    wInProcessData_3: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    wOutProcessData_2: WORD;
    wOutProcessData_3: WORD;
    rNormalizedValue_0: REAL;
    rNormalizedValue_1: REAL;
    enDiagCode_0: IL_INLN_DIAG_CODE;
    enDiagCode_1: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_AI4IU.ChannelConfig[0].InputRange := IL_INLN_AI_RANGE.UNIPOLAR10V;
 fbIL_AI4IU.ChannelConfig[0].NoOfSamples := BYTE#16;
 fbIL_AI4IU.ChannelConfig[0].HighLimit := REAL#10.0;
 fbIL_AI4IU.ChannelConfig[0].LowLimit := REAL#0.0;
 fbIL_AI4IU.ChannelConfig[1].InputRange := IL_INLN_AI_RANGE.BIPOLAR10V;
 fbIL_AI4IU.ChannelConfig[1].NoOfSamples := BYTE#16;
 fbIL_AI4IU.ChannelConfig[1].HighLimit := REAL#10.0;
 fbIL_AI4IU.ChannelConfig[1].LowLimit := REAL#-10.0;
 fbIL_AI4IU.ChannelSelect := BYTE#2#00000011;
 fbIL_AI4IU.InProcData_0 := wInProcessData_0;
 fbIL_AI4IU.InProcData_1 := wInProcessData_1;
 fbIL_AI4IU.InProcData_2 := wInProcessData_2;
 fbIL_AI4IU.InProcData_3 := wInProcessData_3;
 wOutProcessData_0 := fbIL_AI4IU.OutProcData_0;
 wOutProcessData_1 := fbIL_AI4IU.OutProcData_1;
 wOutProcessData_2 := fbIL_AI4IU.OutProcData_2;
 wOutProcessData_3 := fbIL_AI4IU.OutProcData_3;

 // FB call
 fbIL_AI4IU(Enable:=TRUE);

 // Read data
IF fbIL_AI4IU.InOperation = TRUE THEN
 rNormalizedValue_0 := fbIL_AI4IU.OutputData[0].NormalizedValue;
 rNormalizedValue_1 := fbIL_AI4IU.OutputData[1].NormalizedValue;
 enDiagCode_0 := fbIL_AI4IU.OutputData[0].DiagCode;
```

```
 enDiagCode_1 := fbIL_AI4IU.OutputData[1].DiagCode;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_AI4IU".

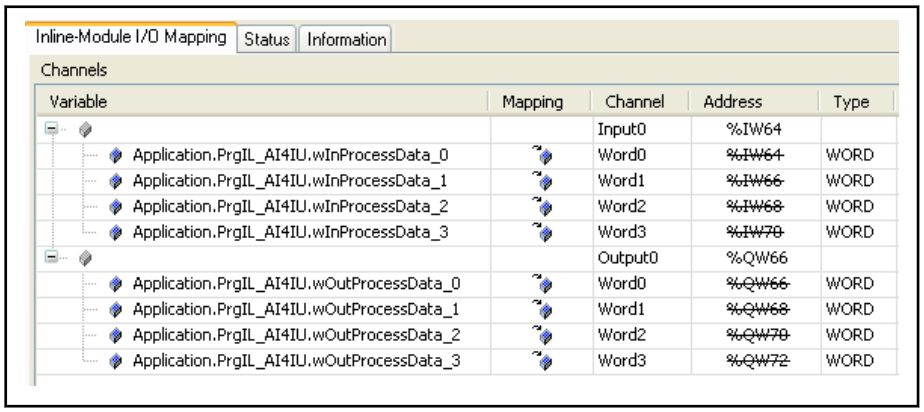


Fig. 14-6: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003004	Channel number	The "InputRange" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003005	Channel number	The "NoOfSamples" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003009	Channel number	Invalid range limits HighLimit <= LowLimit
DEVICE_ERROR (16#0008)	16#00003020	16#000x8010	Invalid configuration

Tab. 14-25: Error codes of the IL_AI4IU

14.2.4 IL_AI8SfIs

Brief description

The **IL_AI8SfIs** function block configures the Inline terminals **R-IB IL AI8/SF** and **R-IB IL AI8/IS** with eight analog input channels and normalizes the data read by the terminal with regard to the user-defined limit values.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus and Profinet IO field bus.

Interface description

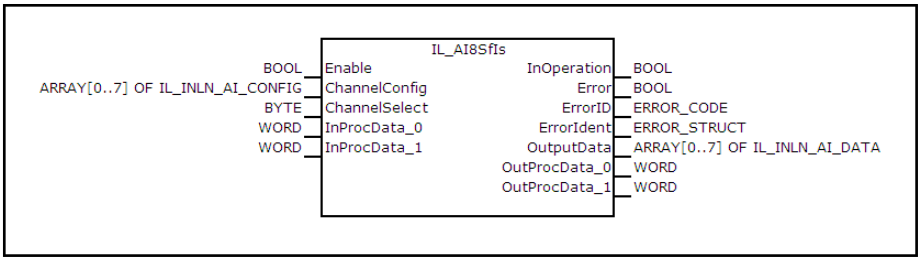


Fig. 14-7: IL_AI8SfIs function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	ARRAY[0..7] OF IL_INLN_AI_CONFIG	Contains configuration data for each individual channel, see "IL_INLN_AI_CONFIG" on page 575
	ChannelSelect	BYTE	Channel selection, Byte#2#11111111; Automatic channel reading is individually activated (bit 0... bit 7 correspond to channel 1...8)
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 577
	OutputData	ARRAY[0..7] OF IL_INLN_AI_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_AI_DATA" on page 575
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-26: IL_AI8SfIs interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	1	255	255	Positive edge at "Enable"

Tab. 14-27: Input behavior of the IL_AI8SfIs

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_AI_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values or diagnostic information of the channels 1 ... 8 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and up to bit 7 to channel 8.

All channels are selected by default.

The measured values are normalized according to the user-defined range limits and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

R-IB IL AI8/SF and R-IB IL AI8/IS operate multiplex (data is read multiplex). Updating data of all channels requires thus more time than one PLC cycle.

IL_INLN_AI_CONFIG

The structure contains information on one channel configuration. The channel is defined by **HighLimit**, **LowLimit**, **NoOfSamples** and **InputRange**.

It is configured as follows:

```
ParameterName[Channel Number].[Function].
```

e.g. to configure **InputRange** when calling channel 1 for the first time:

```
ChannelConfig[0].InputRange.
```

Name	Type	Description	Min. value	Max. value	Default value	Effective
HighLimit	REAL	Upper limit of the normalized value	-65535.0	65535.0	10.0	Continuously with "Enable"
LowLimit	REAL	Lower limit of the normalized value	-65535.0	65535.0	0.0	Continuously with "Enable"
NoOfSamples	BYTE	0 – No average value generation 4 – Average value generation using 4 measured values 16 – Average value generation using 16 measured values 32 – Average value generation using 32 measured values	0	32	16	Positive edge at "Enable"
InputRange	IL_INLN_AI_RANGE	Measuring range, see "IL_INLN_AI_RANGE" on page 576	UNIPOLAR10V	BIPOLAR40MA	UNIPOLAR10V	Positive edge at "Enable"

Tab. 14-28: Permitted values of IL_INLN_AI_CONFIG for IL_AI8Sfls

IL_INLN_AI_DATA

The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

```
ParameterName[Channel_Number].[Function]
```

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value between upper and lower limit value
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 576

Tab. 14-29: Structural elements IL_INLN_AI_DATA

IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-30: Enum IL_INLN_DIAG_CODE

IL_INLN_AI_RANGE

The enumeration contains the measuring ranges to configure one channel.

ENUM	Description
UNIPOLAR10V	Voltage measurement 0 ... 10 V
BIPOLAR10V	Voltage measurement -10 V ... +10 V
UNIPOLAR5V	Voltage measurement 0 … 5 V
BIPOLAR5V	Voltage measurement -5 V ... +5 V
UNIPOLAR25V	Voltage measurement 0 ... 25 V
BIPOLAR25V	Voltage measurement -25 V ... +25 V
UNIPOLAR50V	Voltage measurement 0 ... 50 V
UNIPOLAR20MA	Current measurement 0 ... 20 mA
BIPOLAR20MA	Current measurement -20 mA .... +20 mA
UNIPOLAR4TO20MA	Current measurement 4 ... 20 mA
UNIPOLAR40MA	Current measurement 0 ... 40 mA
BIPOLAR40MA	Current measurement -40 mA .... +40 mA

Tab. 14-31: Permitted values of IL_INLN_AI_RANGE for IL_AI8Sfls

Example

In the following example, channel 1 is configured for a range between 0 V and +10 V and channel 2 for the range 0... +5V. Data is located between the upper and lower limit and available as normalized variable values.

Declaration of the example

```
PROGRAM PrgIL_AI8SfIs
VAR
    fbIL_AI8SfIs: IL_AI8SfIs;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_1: WORD;
    wOutProcessData_0: WORD;
    rNormValCH1: REAL;
    rNormValCH2: REAL;
    enDiagCodeCH1: IL_INLN_DIAG_CODE;
    enDiagCodeCH2: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_AI8SfIs.ChannelConfig[0].HighLimit := REAL#10.0;
 fbIL_AI8SfIs.ChannelConfig[0].InputRange := IL_INLN_AI_RANGE.UNIPOLAR10V;
 fbIL_AI8SfIs.ChannelConfig[0].LowLimit := REAL#0.0;
 fbIL_AI8SfIs.ChannelConfig[0].NoOfSamples := BYTE#16;
 fbIL_AI8SfIs.ChannelConfig[1].HighLimit := REAL#5.0;
 fbIL_AI8SfIs.ChannelConfig[1].InputRange := IL_INLN_AI_RANGE.UNIPOLAR5V;
 fbIL_AI8SfIs.ChannelConfig[1].LowLimit := REAL#0.0;
 fbIL_AI8SfIs.ChannelConfig[1].NoOfSamples := BYTE#32;
 fbIL_AI8SfIs.ChannelSelect := BYTE#2#00000011;
 fbIL_AI8SfIs.InProcData_0 := wInProcessData_0;
 fbIL_AI8SfIs.InProcData_1 := wInProcessData_1;
 wOutProcessData_0 := fbIL_AI8SfIs.OutProcData_0;
 wOutProcessData_1 := fbIL_AI8SfIs.OutProcData_1;

 // FB call
  fbIL_AI8SfIs(Enable :=TRUE);

 // Read data
 IF fbIL_AI8SfIs.InOperation = TRUE THEN
  rNormValCH1 := fbIL_AI8SfIs.OutputData[0].NormalizedValue;
  rNormValCH2 := fbIL_AI8SfIs.OutputData[1].NormalizedValue;
  enDiagCodeCH1 := fbIL_AI8SfIs.OutputData[0].DiagCode;
  enDiagCodeCH2 := fbIL_AI8SfIs.OutputData[1].DiagCode;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_AI8Sfls".

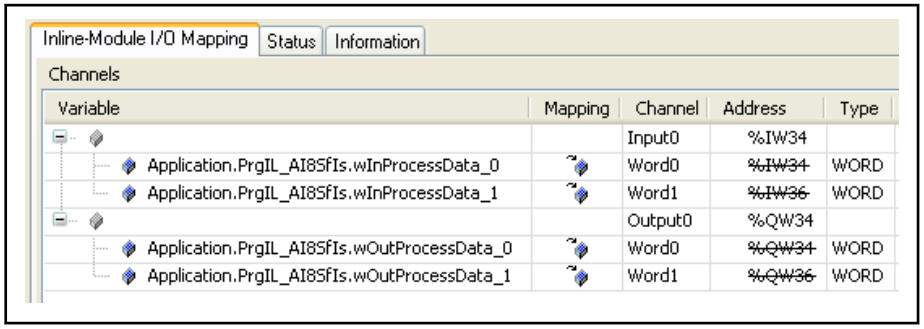


Fig. 14-8: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003004	Channel number	The "InputRange" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003005	Channel number	The "NoOfSamples" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003009	Channel number	Invalid range limits HighLimit <= LowLimit
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#00003020	16#000x_8010	Invalid configuration

Tab. 14-32: Error codes of the IL_AI8Sfls

14.2.5 IL_Sgi2f

Brief description

The **IL_Sgi2f** function block initializes and configures the Inline terminal **R-IB IL SGI 2/F-PAC** with two analog inputs for strain gauges and normalizes data according to the user-defined nominal load.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus and Profinet IO field bus.

Interface description

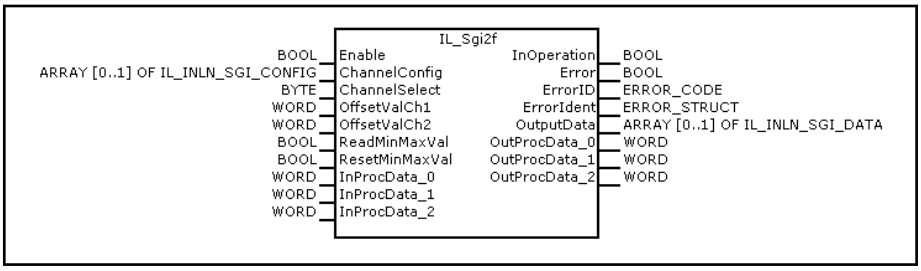


Fig. 14-9: IL_Sgi2f function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	ARRAY[0..1] OF IL_INLN_SGI_CONFIG	Contains configuration data for each individual channel, see "IL_INLN_SGI_CONFIG" on page 580
	ChannelSelect	BYTE	Byte#2#0000_0011; channel selection Bit0- Channel 1, Bit1- Channel 2
	OffsetValCh1	WORD	Offset for channel 1
	OffsetValCh2	WORD	Offset for channel 2
	ReadMinMaxVal	BOOL	Allows reading min./max. values for the selected channels
	ResetMinMaxVal	BOOL	Allows resetting min./max. values for the selected channels
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
	InProcData_2	WORD	Process input word 2
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 583
	OutputData	ARRAY[0..1] OF IL_INLN_SGI_DATA	Contains the minimum, maximum and normalized value and the diagnostic code for each channel, see "IL_INLN_SGI_DATA" on page 581
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1
	OutProcData_2	WORD	Process output word 2

Tab. 14-33: IL_Sgi2f interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	1	3	3	Positive edge at "Enable"
ReadMinMaxVal	BOOL	FALSE	TRUE	FALSE	Continuously with "Enable"
ResetMinMaxVal	BOOL	FALSE	TRUE	FALSE	Continuously with "Enable"

Tab. 14-34: Input behavior of the IL_Sgi2f

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_SGI_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values of channel 1 and 2 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and bit 1 to channel 2.

All channels are selected by default.

The measured values are normalized according to the user-defined nominal load and displayed in **NormalizedValue**.

If an error occurs, it is displayed for the respective channel in **DiagCode**.

The minimum and maximum values of a channel are displayed after triggering **ReadminMaxVal**. **ResetMinMAxVal** sets the min./max. values to zero.

1. If **ReadMinMaxVal** or **ResetMinMaxVal** was triggered, the time to calculate a **NormalizedValue** is more than one PLC cycle.
2. If the range is exceeded while reading "sgi" data, the minimum value of the "sgi" is used instead of the value with range exceedance. This value is kept until the user resets the minimum value manually.

IL_INLN_SGI_CONFIG

The structure supports the configuration of one channel.

The channel can be configured with regard to **Filter**, **InputRange**, **NominalLoad** and **Offset**.

It is configured as follows:

```
ParameterName[Channel_Number].[Function].
```

e.g. to configure **InputRange** when calling channel 1 for the first time:

```
ChannelConfig[0].InputRange.
```

Name	Type	Description	Min. value	Max. value	Default value	Effective
Filter	BOOL	Filter selection TRUE - 16 times average value generation FALSE - No average value generation	FALSE	TRUE	TRUE	Positive edge at "Enable"
InputRange	IL_INLN_SGI_RANGE	Input range	UNIPOLAR1MV	BIPOLAR4MV	UNIPOLAR2MV	Positive edge at "Enable"

Name	Type	Description	Min. value	Max. value	Default value	Effective
NominalLoad	REAL	Nominal load (Should be >0)	-----	-----	0	Positive edge at "Enable"
Offset	WORD	Offset in strain gauge	-----	-----	0	Continuous

Input range, see "IL_INLN_SGI_RANGE" on page 581

Use the function block IL_Sgi2fOffset, page 584, to determine the offset values for the corresponding channel of the device.

IL_INLN_SGI_DATA

The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value in corresponding nominal load
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code
MinValue	REAL	Minimum value read from strain gauge
MaxValue	REAL	Maximum value read from strain gauge

Diagnostic code, see "IL_INLN_DIAG_CODE" on page 581

IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-35: Enum IL_INLN_DIAG_CODE

IL_INLN_SGI_RANGE

The enumeration contains the measuring ranges to configure one channel.

ENUM	Description
UNIPOLAR1MV	+1 mV/V
BIPOLAR1MV	±1 mV/V
UNIPOLAR2MV	+2 mV/V
BIPOLAR2MV	±2 mV/V

ENUM	Description
UNIPOLAR3MV	+3 mV/V
BIPOLAR3MV	±3 mV/V
UNIPOLAR4MV	+4 mV/V
BIPOLAR4MV	±4 mV/V

Example

The function block is used to configure an Inline terminal with two analog inputs for strain gauges suitable to connect load cells, force transducers, melt pressure sensors and similar based on strain gauges.

Configured are channel selection, offset values and input ranges. Process data is also mapped.

In this example. the function block **Sgi2f** is preset to +2 mV/V for channel 1 and to +1 mV/V for channel 2.

Read data is scaled with regard to its nominal load and it is available in **rNormValCH1** and **rNormValCH2** for the respective channel.

Use the function block IL_Sgi2fOffset, page 584, to determine the offset values for the corresponding channel of the device.

Declaration of the example

```
PROGRAM PrgIL_Sgi2f
VAR
    fbIL_Sgi2f: IL_Sgi2f;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wInProcessData_2: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    wOutProcessData_2: WORD;
    fbIL_Sgi2fOffsetCH1: IL_Sgi2fOffset;
    fbIL_Sgi2fOffsetCH2: IL_Sgi2fOffset;
    rNormValCH1: REAL;
    rNormValCH2: REAL;
    enDiagCodeCH1: IL_INLN_DIAG_CODE;
    enDiagCodeCH2: IL_INLN_DIAG_CODE;
    rMinValCH1: REAL;
    rMinValCH2: REAL;
    rMaxValCH1: REAL;
    rMaxValCH2: REAL;
END_VAR
```

Implementation of the example

```
// Offset configuration
 fbIL_Sgi2fOffsetCH1.InputData := wInProcessData_1;
 fbIL_Sgi2fOffsetCH1.NoOfSamples := INT#16;
 fbIL_Sgi2fOffsetCH1(Calibrate := TRUE);
 fbIL_Sgi2fOffsetCH2.InputData := wInProcessData_2;
 fbIL_Sgi2fOffsetCH2.NoOfSamples := INT#16;
 fbIL_Sgi2fOffsetCH2(Calibrate := TRUE);
// Configuration
 fbIL_Sgi2f.ChannelConfig[0].Filter := TRUE;
 fbIL_Sgi2f.ChannelConfig[0].InputRange := IL_INLN_SGI_RANGE.UNIPOLAR2MV;
 fbIL_Sgi2f.ChannelConfig[0].NominalLoad := REAL#60;
 fbIL_Sgi2f.ChannelConfig[1].Filter := FALSE;
 fbIL_Sgi2f.ChannelConfig[1].InputRange := IL_INLN_SGI_RANGE.UNIPOLAR1MV;
 fbIL_Sgi2f.ChannelConfig[1].NominalLoad := REAL#100;
 fbIL_Sgi2f.ReadMinMaxVal := FALSE;
 fbIL_Sgi2f.ResetMinMAxVal := FALSE;
 fbIL_Sgi2f.InProcData_0 := wInProcessData_0;
 fbIL_Sgi2f.InProcData_1 := wInProcessData_1;
 fbIL_Sgi2f.InProcData_2 := wInProcessData_2;
 wOutProcessData_0 := fbIL_Sgi2f.OutProcData_0;
 wOutProcessData_1 := fbIL_Sgi2f.OutProcData_1;
 wOutProcessData_2 := fbIL_Sgi2f.OutProcData_2;
 fbIL_Sgi2f.ChannelSelect := BYTE#2#00000011;
 fbIL_Sgi2f.OffsetValCh1 := fbIL_Sgi2fOffsetCH1.OffsetVal;
```

```
 fbIL_Sgi2f.OffsetValCh2 := fbIL_Sgi2fOffsetCH2.OffsetVal;

 // FB call
 fbIL_Sgi2f(Enable := TRUE);

 // Read data
  IF fbIL_Sgi2f.InOperation = TRUE THEN
  fbIL_Sgi2f.ReadMinMaxVal := FALSE;
  fbIL_Sgi2f.ResetMinMAxVal := FALSE;
  rNormValCH1 := fbIL_Sgi2f.OutputData[0].NormalizedValue;
  rNormValCH2 := fbIL_Sgi2f.OutputData[1].NormalizedValue;
  enDiagCodeCH1 := fbIL_Sgi2f.OutputData[0].DiagCode;
  enDiagCodeCH2 := fbIL_Sgi2f.OutputData[1].DiagCode;
 END_IF
// Read min. and max. values
 IF fbIL_Sgi2f.InOperation = TRUE THEN
  fbIL_Sgi2f.ReadMinMaxVal := TRUE;
  fbIL_Sgi2f.ResetMinMAxVal := FALSE;
  rMinValCH1 := fbIL_Sgi2f.OutputData[0].MinimumValue;
  rMinValCH2 := fbIL_Sgi2f.OutputData[1].MinimumValue;
  rMaxValCH1 := fbIL_Sgi2f.OutputData[0].MaximumValue;
  rMaxValCH2 := fbIL_Sgi2f.OutputData[1].MaximumValue;
 END_IF
// Reset min. and max. values
  IF fbIL_Sgi2f.InOperation = TRUE THEN
  fbIL_Sgi2f.ReadMinMaxVal := FALSE;
  fbIL_Sgi2f.ResetMinMAxVal := TRUE;
  rMinValCH1 := fbIL_Sgi2f.OutputData[0].MinimumValue;
  rMinValCH2 := fbIL_Sgi2f.OutputData[1].MinimumValue;
  rMaxValCH1 := fbIL_Sgi2f.OutputData[0].MaximumValue;
  rMaxValCH2 := fbIL_Sgi2f.OutputData[1].MaximumValue;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and wOutProcessData_x are declared in the program "prgIL_Sgi2f".

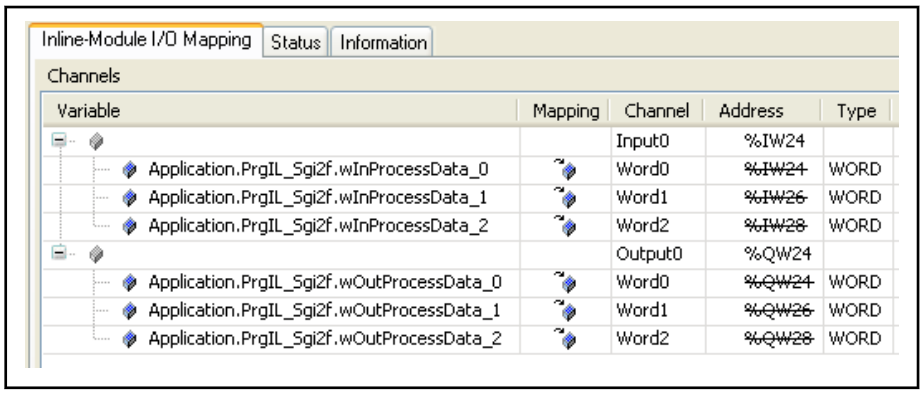


Fig. 14-10: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	“ChannelSelect” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003004	Channel number	The "InputRange" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003015	Channel number	“NominalLoad" of a channel is invalid

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#00003020	16#000x8010	Invalid configuration *Possible causes:* 1. The jumper voltage is not 3.3 V ± 0.5 V or 5 V ± 0.5 V 2. The characteristic value is ±3 mV/V or ±4 mV/V and the jumper voltage is 5 V ±0.5 V. Change jumper voltage to 3.3 V ±0.5 V

Tab. 14-36: Error codes of the IL_Sgi2f

14.2.6 IL_Sgi2fOffset

Brief description

The **IL_Sgi2fOffset** function block can be used to calibrate the Inline terminal **R-IB IL SGI 2/F-PAC** (Inline terminal with two analog inputs for strain gauges).

After a positive edge at the **Calibrate** input, the function block computes the average value of the user-defined number of samples.

Interface description

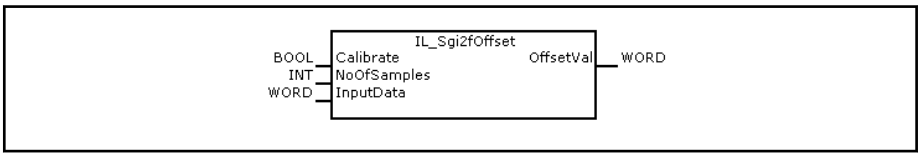


Fig. 14-11: IL_Sgi2fOffset function block

	Name	Type	Description
VAR_INPUT	Calibrate	BOOL	Positive edge starts calculation
	NoOfSamples	INT	Number of samples to be read (1...32)
	InputData	WORD	Input data
VAR OUTPUT	OffsetVal	WORD	Offset

Tab. 14-37: IL_Sgi2fOffset interface

For a use case, refer to the description of the function block IL_Sgi2f, page 578,.

14.2.7 IL_Temp2Uth

Brief description

The **IL_Temp2Uth** function block configures the Inline terminal **R-IB IL TEMP 2 UTH-PAC** with two analog inputs for thermocouples and normalizes data according to the user-defined resolution.

Compatibility

The function block can be used for Inline bus and Profinet IO field bus.

Interface description

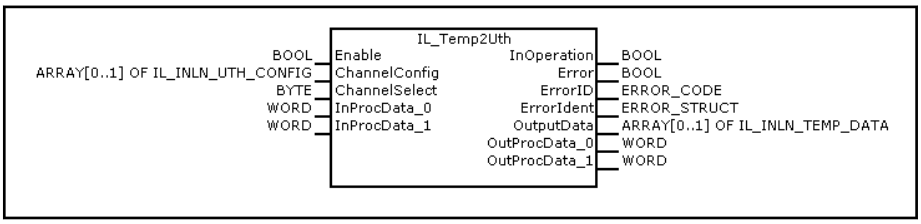


Fig. 14-12: IL_Temp2Uth function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	ARRAY[0..1] OF IL_INLN_UTH_CONFIG	Contains configuration data for the channels 1...2, see "IL_INLN_UTH_CONFIG" on page 586
	ChannelSelect	BYTE	Byte#2#0000_0011; Individual channel activation (Bit 0 - channel 1, bit 1 - channel 2)
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 589
	OutputData	ARRAY[0..1] OF IL_INLN_TEMP_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_TEMP_DATA" on page 586
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-38: IL_Temp2Uth interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	0	3	3	Positive edge at "Enable"

Tab. 14-39: Input behavior of the IL_Temp2Uth

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_UTH_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values or diagnostic information of the channels 1 ... 2 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and bit 1 to channel 2.

All channels are selected by default.

The measured values are normalized according to the user-defined resolution and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

IL_INLN_UTH_CONFIG

The structure contains information on one channel configuration. The channel is defined by **Resolution**, **SensorType** and **CJCompensation**.

It is configured as follows:

`ParameterName[Channel_Number].[Function].`

e.g. to configure **SensorType** when calling channel 1 for the first time:

`ChannelConfig[0].SensorType.`

Name	Type	Description	Min. value	Max. value	Default value	Effective
Resolution	WORD	Triggering: **0 : 0.1°C** 1 : 1°C 2 : 0.1°F 3 : 1°F See table "Resolution" below	**0**	3	**0**	Positive edge at "Enable"
SensorType	IL_INLN_UTH_SENSOR	Sensor type selection, see "IL_INLN_UTH_SENSOR" on page 587 Example: ETYPE	**KTYPE**	VOLT	**KTYPE**	Positive edge at "Enable"
CJCompensation	IL_INLN_UTH_CJCOMP	Selection of cold junction compensation, see "IL_INLN_UTH_CJCOMP" on page 587	INTERNALCJ_ACTIVE	EXTERNALCJ_NI1000	INTERNALCJ_ACTIVE	Positive edge at "Enable"

Resolution

Decimal	Binary	Description
0	00	Resolution: 0,1°C (1 µV)
1	01	Resolution: 1°C (10 µV)
2	10	Resolution: 0.1°F
3	11	Resolution: 1°F

IL_INLN_TEMP_DATA

The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value in corresponding resolution
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 587

Tab. 14-40: Structural elements IL_INLN_TEMP_DATA

IL_INLN_DIAG_CODE The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-41: Enum IL_INLN_DIAG_CODE

IL_INLN_UTH_CJCOMP The enumeration selects the cold junction compensation.

ENUM	Description
INTERNALCJ_ACTIVE	**Internal cold junction active;** TC measurement with internal cold junction compensation
INTERNALCJ_INACTIVE	Internal cold junction inactive; TC differential measurement without cold junction compensation
EXTERNALCJ_PT1000	External cold junction Pt1000; TC measurement with external cold junction compensation at an isothermal block **(For lab applications only)**
EXTERNALCJ_NI1000	External cold junction Ni1000; TC measurement with external cold junction compensation at an isothermal block **(For lab applications only)**

IL_INLN_UTH_SENSOR The enumeration type is used for the sensor type selection for the individual channels.

ENUM	Description
KTYPE	**ThermocoupleType - K sensor**
JTYPE	ThermocoupleType - J sensor
ETYPE	ThermocoupleType - E sensor
RTYPE	ThermocoupleType - R sensor
STYPE	ThermocoupleType - S sensor
TTYPE	ThermocoupleType - T sensor
BTYPE	ThermocoupleType - B sensor
NTYPE	ThermocoupleType - N sensor
UTYPE	ThermocoupleType - U sensor
LTYPE	ThermocoupleType - L sensor
CTYPE	ThermocoupleType - C sensor
WTYPE	ThermocoupleType - W sensor

ENUM	Description
HKTYPE	ThermocoupleType - HK sensor
COLDJUNCTION	Cold junction; CJ
VOLT	U: Voltage (-15 mV ... +85 mV)

Example The function block configures a UTH temperature terminal for the temperature measurement.

Make the following configuration presettings: Configuration of inputs, channel selection and process data mapping to the device.

In this example, Temp2Uth is configured as follows:

Channel 1: ‘KTYPE’ and channel 2: ‘JTYPE’.

To read data, the "NormalizedValue" of the output structure is available for each channel.

Declaration of the example

```
PROGRAM PrgIL_Temp2Uth
VAR
    fbIL_Temp2Uth: IL_Temp2Uth;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    rNormValCH1: REAL;
    rNormValCH2: REAL;
    enDiagCodeCH1: IL_INLN_DIAG_CODE;
    enDiagCodeCH2: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_Temp2Uth.ChannelConfig[0].CJCompensation := IL_INLN_UTH_CJCOMP.INTERNALCJ_ACTIVE;
 fbIL_Temp2Uth.ChannelConfig[0].Resolution     := WORD#0;
 fbIL_Temp2Uth.ChannelConfig[0].SensorType     := IL_INLN_UTH_SENSOR.KTYPE;
 fbIL_Temp2Uth.ChannelConfig[1].CJCompensation := IL_INLN_UTH_CJCOMP.INTERNALCJ_INACTIVE;
 fbIL_Temp2Uth.ChannelConfig[1].Resolution     := WORD#1;
 fbIL_Temp2Uth.ChannelConfig[1].SensorType     := IL_INLN_UTH_SENSOR.JTYPE;
 fbIL_Temp2Uth.ChannelSelect                   := BYTE#2#00000011;
 fbIL_Temp2Uth.InProcData_0                    := wInProcessData_0;
 fbIL_Temp2Uth.InProcData_1                    := wInProcessData_1;
 wOutProcessData_0                                    := fbIL_Temp2Uth.OutProcData_0;
 wOutProcessData_1                                    := fbIL_Temp2Uth.OutProcData_1;

 // FB call
 fbIL_Temp2Uth(Enable := TRUE);

 // Data read
IF fbIL_Temp2Uth.InOperation = TRUE THEN
    rNormValCH1         := fbIL_Temp2Uth.OutputData[0].NormalizedValue;
    rNormValCH2         := fbIL_Temp2Uth.OutputData[1].NormalizedValue;
    enDiagCodeCH1    := fbIL_Temp2Uth.OutputData[0].DiagCode;
    enDiagCodeCH2    := fbIL_Temp2Uth.OutputData[1].DiagCode;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_Temp2Uth".

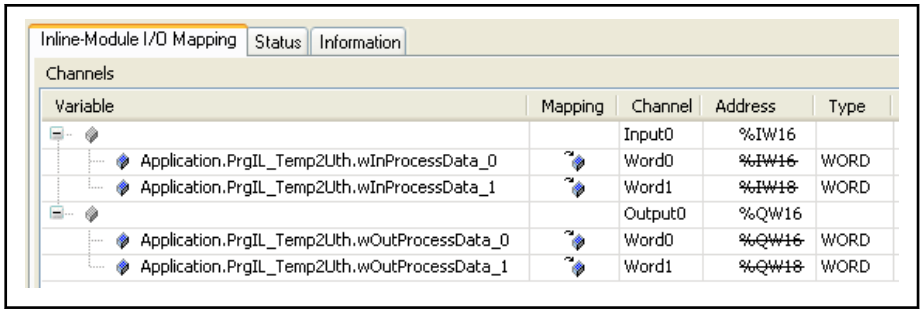


Fig. 14-13: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003007	Channel number	The "Resolution" input of a channel is invalid or does not match the resolution of the sensor type
INPUT_INVALID_ERROR (16#0001)	16#00003008	Channel number	"SensorType" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003016	Channel number	The ‘CodeJunctionCompensation' is invalid

Tab. 14-42: Error codes of the IL_Temp2Uth

14.2.8 IL_Temp2Rtd

Brief description

The **IL_Temp2Rtd** function block initializes and configures the Inline terminal **R-IB IL TEMP2 RTD-PAC** with two analog inputs for temperature sensors and normalizes data according to the user-defined nominal resolution.

Compatibility

The function block can be used for Inline bus and Profinet IO field bus.

Interface description

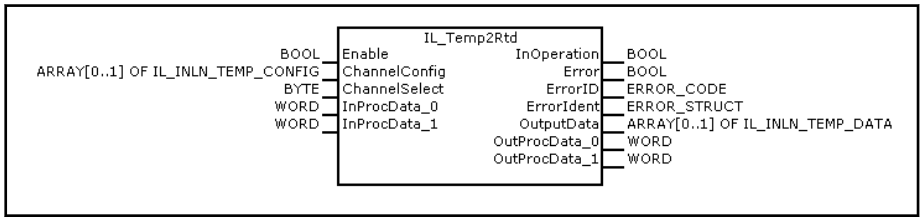


Fig. 14-14: IL_Temp2Rtd function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	ARRAY[0..1] OF IL_INLN_TEMP_CONFIG	Contains configuration data for the channels 1...2, see "IL_INLN_TEMP_CONFIG" on page 590
	ChannelSelect	BYTE	Byte#2#0000_0011; Individual channel activation (Bit 0 - channel 1, bit 1 - channel 2)
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 594
	OutputData	ARRAY[0..1] OF IL_INLN_TEMP_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_TEMP_DATA" on page 591
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-43: IL_Temp2Rtd interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	1	3	3	Positive edge at "Enable"

Tab. 14-44: Input behavior of the IL_Temp2Rtd

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_AI_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values or diagnostic information of the channels 1 ... 2 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and bit 1 to channel 2.

All channels are selected by default.

The measured values are normalized according to the user-defined resolution and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

IL_INLN_TEMP_CONFIG

The structure contains information on one channel configuration. The channel is defined by **SensorType**, **ConnectionType**, **NoOfSamples**, **Resolution** and **Resistance**.

It is configured as follows:

```
ParameterName[Channel_Number].[Function].
```

e.g. to configure **SensorType** when calling channel 1 for the first time:

`ChannelConfig[0].SensorType`.

Name	Type	Description	Min. value	Max. value	Default value	Effective
NoOfSamples	BYTE	Only used for IL_Temp4_8Rtd				
SensorType	IL_INLN_TEMP_SENSOR	Sensor type selection, see "IL_INLN_TEMP_SENSOR" on page 593 Example: PTSAMA	PTDIN	LINEAR4000	PTDIN	Positive edge at "Enable"
Connection-Type	IL_INLN_TEMP_CONNECTIONTYPE	Sensor connection: THREEWIRE := 3-wire TWOWIRE := 2-wire FOURWIRE := 4-wire See "IL_INLN_TEMP_CONNECTIONTYPE" on page 593	THREE-WIRE	FOUR-WIRE	THREE-WIRE	Positive edge at "Enable"
Resolution	WORD	Resolution: 0 : 0.1°C/1%/0.1 Ω 1 : 0.01°C/0.1%/0.01Ω 2 : 0.1°F/ reserved 3 : 0.01°F/ reserved Refer to the resolution table below	0	3	0	Positive edge at "Enable"
Resistance	IL_INLN_TEMP_R0	Resistance reference value R0 [Ω], see "IL_INLN_TEMP_R0" on page 592	R10	R3000	R100	Positive edge at "Enable"

Tab. 14-45: Permitted values of IL_INLN_AI_CONFIG for IL_Temp2Rtd

Resolution table

Code		Resolution for sensor type			
Dec.	Bin.	0 ...11	13 (pot)%	14 (Linear400)	16(Linear4000)
0	00	0.1°C	1%	0.1 Ω	1 Ω
1	01	0.01°C	0.1%	0.01 Ω	0.1 Ω
2	10	0.1°F	Reserved	Reserved	Reserved
3	11	0.01°F	Reserved	Reserved	Reserved

IL_INLN_TEMP_DATA

The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value in corresponding resolution
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 592

Tab. 14-46: Structural elements IL_INLN_TEMP_DATA

IL_INLN_DIAG_CODE The table contains channel-related diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-47: Enum IL_INLN_DIAG_CODE

IL_INLN_TEMP_R0 The resistance reference value to configure the channel can be selected using this enumeration type.

Enum	Description
R10	Resistance reference value R0: 10 Ω
R20	Resistance reference value R0: 20 Ω
R30	Resistance reference value R0: 30 Ω
R50	Resistance reference value R0: 50 Ω
R100	Resistance reference value R0: 100 Ω
R120	Resistance reference value R0: 120 Ω
R150	Resistance reference value R0: 150 Ω
R200	Resistance reference value R0: 200 Ω
R240	Resistance reference value R0: 240 Ω
R300	Resistance reference value R0: 300 Ω
R400	Resistance reference value R0: 400 Ω
R500	Resistance reference value R0: 500 Ω
R1000	Resistor reference value R0: 1000 Ω
R1500	Resistor reference value R0: 1500 Ω
R2000	Resistance reference value R0: 2000 Ω

Enum	Description
R10000	Only for the IL_Temp4_8Rtd function block
R3000	Resistor reference value R0: 3000 Ω

Tab. 14-48: Enum IL_INLN_TEMP_R0 for IL_Temp2Rtd

IL_INLN_TEMP_SENSOR

The sensor type to configure a channel can be selected using this enumeration type.

Enum	Sensor type
PTDIN	PT DIN
PTSAMA	PT SAMA
NIDIN	NI DIN
NISAMA	NI SAMA
CU10	CU10
CU50	CU50
CU53	CU53
NI1000	NI1000 (Landis & Gyr)
NI500	NI500 (Viesmann)
KTY81110	KTY81-100
KTY84	KTY84
KTY81210	KTY81210
POT	POTENTIOMETER (%)
LINEAR400	Linear R: 0 ... 400 Ω
LINEAR4000	Linear R:0 ... 4000 Ω

Tab. 14-49: Enum IL_INLN_TEMP_SENSOR for the IL_Temp2Rtd function block

IL_INLN_TEMP_CONNECTION-TYPE

The connection type of the sensor can be selected using this enumeration type.

Enum	Description
THREEWIRE	Connection type: 3-wire sensor
TWOWIRE	Connection type: 2-wire sensor
FOURWIRE	Connection type: 4-wire sensor

Tab. 14-50: Enum IL_INLN_TEMP_CONNECTIONTYPE for IL_Temp2Rtd

Example

The function block configures an RTD temperature terminal for the temperature measurement.

Make the following configuration presettings: Configuration of inputs, channel selection and process data mapping to the device.

In this example, Temp2Rtd is configured as follows:

Channel 1: ‘PTDIN’ and channel 2: ‘POT’.

To read data, the "NormalizedValue" of the output structure is available for each channel.

Declaration of the example

```
PROGRAM PrgIL_Temp2Rtd
VAR
    fbIL_Temp2Rtd: IL_Temp2Rtd;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    rNormValCH1: REAL;
    rNormValCH2: REAL;
    enDiagCodeCH1: IL_INLN_DIAG_CODE;
    enDiagCodeCH2: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_Temp2Rtd.ChannelConfig[0].ConnectionType := IL_INLN_TEMP_CONNECTIONTYPE.THREEWIRE;
 fbIL_Temp2Rtd.ChannelConfig[0].NoOfSamples    := BYTE#16;
 fbIL_Temp2Rtd.ChannelConfig[0].Resistance     := IL_INLN_TEMP_R0.R100;
 fbIL_Temp2Rtd.ChannelConfig[0].Resolution     := WORD#1;
 fbIL_Temp2Rtd.ChannelConfig[0].SensorType     := IL_INLN_TEMP_SENSOR.PTDIN;
 fbIL_Temp2Rtd.ChannelConfig[1].ConnectionType := IL_INLN_TEMP_CONNECTIONTYPE.TWOWIRE;
 fbIL_Temp2Rtd.ChannelConfig[1].NoOfSamples    := BYTE#32;
 fbIL_Temp2Rtd.ChannelConfig[1].Resistance     := IL_INLN_TEMP_R0.R1000;
 fbIL_Temp2Rtd.ChannelConfig[1].Resolution     := WORD#0;
 fbIL_Temp2Rtd.ChannelConfig[1].SensorType     := IL_INLN_TEMP_SENSOR.POT;
 fbIL_Temp2Rtd.ChannelSelect                   := BYTE#2#00000011;
 fbIL_Temp2Rtd.InProcData_0                    := wInProcessData_0;
 fbIL_Temp2Rtd.InProcData_1                    := wInProcessData_1;
 wOutProcessData_0                            := fbIL_Temp2Rtd.OutProcData_0;
 wOutProcessData_1                            := fbIL_Temp2Rtd.OutProcData_1;

 // FB call
 fbIL_Temp2Rtd(Enable:= TRUE );

 // Data read
 IF fbIL_Temp2Rtd.InOperation = TRUE THEN
     rNormValCH1    := fbIL_Temp2Rtd.OutputData[0].NormalizedValue;
     rNormValCH2    := fbIL_Temp2Rtd.OutputData[1].NormalizedValue;
     enDiagCodeCH1  := fbIL_Temp2Rtd.OutputData[0].DiagCode;
     enDiagCodeCH2  := fbIL_Temp2Rtd.OutputData[1].DiagCode;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_Temp2Rtd".

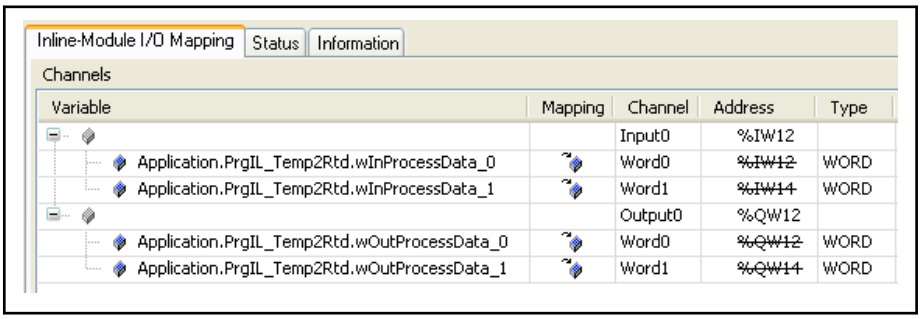


Fig. 14-15: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003006	Channel number	“Resistance” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003007	Channel number	The "Resolution" input of a channel is invalid or does not match the resolution of the sensor type
INPUT_INVALID_ERROR (16#0001)	16#00003008	Channel number	"SensorType" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003014	Channel number	The “ConnectionType" input of a channel is invalid or “FOURWIRE” does not match channel 2.

Tab. 14-51: Error codes of the IL_Temp2Rtd

14.2.9 IL_Temp4_8Rtd

Brief description

The **IL_Temp4_8Rtd** function block initializes and configures the Inline terminal **R-IB IL TEMP 4/8 RTD-PAC** with eight resistive temperature sensor inputs and normalizes data according to the user-defined nominal resolution.

Compatibility

The function block can be used for Inline bus, Profibus and Profinet IO field bus.

Interface description

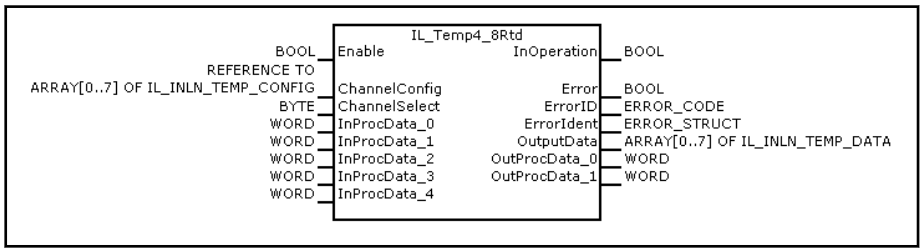


Fig. 14-16: IL_Temp4_8Rtd function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	REFERENCE TO ARRAY[0..7] OF IL_INLN_TEMP_CONFIG	Contains configuration data for the channels 1...8, see "IL_INLN_TEMP_CONFIG" on page 596
	ChannelSelect	BYTE	Byte#2#11111111; Individual channel activation (bit 0... bit 7 correspond to channel 1...8)
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
	InProcData_2	WORD	Process input word 2
	InProcData_3	WORD	Process input word 3
	InProcData_4	WORD	Process input word 4

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 601
	OutputData	ARRAY[0..7] OF IL_INLN_TEMP_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_TEMP_DATA" on page 598
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-52: IL_Temp4_8Rtd interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	1	255	255	Positive edge at "Enable"

Tab. 14-53: Input behavior of the IL_Temp4_8Rtd

Functional Description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_TEMP_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values or diagnostic information of the channels 1 ... 8 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and up to bit 7 to channel 8.

All channels are selected by default.

The measured values are normalized according to the user-defined resolution and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

☞ R-IB IL TEMP4_8RTD-PAC operates in multiplex (data is thus also read in multiplex). Updating data for each cycle thus requires longer than one PLC cycle.

IL_INLN_TEMP_CONFIG

The structure contains information on one channel configuration. The channel is defined by **SensorType**, **ConnectionType**, **NoOfSamples**, **Resolution** and **Resistance**.

It is configured as follows:

`ParameterName[Channel_Number].[Function].`

e.g. to configure **SensorType** when calling channel 1 for the first time:

`ChannelConfig[0].SensorType.`

Name	Type	Description	Min. value	Max. value	Default value	Effective
NoOfSamples	BYTE	0 – No average value generation 4 – Average value generation using 4 measured values 16 – Average value generation using 16 measured values 32 – Average value generation using 32 measured values	0	32	16	Positive edge at "Enable"
SensorType	IL_INLN_TEMP_SENSOR	Sensor type selection, see "IL_INLN_TEMP_SENSOR" on page 599 Example: PTSAMA	PTDIN	LINEAR20000	PTDIN	Positive edge at "Enable"
Connection-Type	IL_INLN_TEMP_CONNECTIONTYPE	Sensor connection: THREEWIRE := 3-wire TWOWIRE := 2-wire Refer to "IL_INLN_TEMP_CONNECTIONTYPE" on page 599	THREE-WIRE	TWO-WIRE	THREE-WIRE	Positive edge at "Enable"
Resolution	WORD	Resolution: **0 : 0.1°C/1%/0.1 Ω** 1 : 0.01°C/0.1%/0.01Ω 2 : 0.1°F/ reserved 3 : 0.01°F/ reserved Refer to the resolution table below	0	3	0	Positive edge at "Enable"
Resistance	IL_INLN_TEMP_R0	Resistance reference value R0 [Ω], see "IL_INLN_TEMP_R0" on page 598	R10	R10000	R100	Positive edge at "Enable"

Tab. 14-54: Permitted values of IL_INLN_AI_CONFIG for IL_Temp4_8Rtd

Resolution table

Code		Resolution for sensor type			
Dec.	**Bin.**	**0 ...11**	**13 (pot)%**	**14 (Linear400)**	**15 (Linear20000)**
0	00	0.1°C	1%	0.1 Ω	1 Ω
1	01	0.01°C	0.1%	0.01 Ω	0.1 Ω
2	10	0.1°F	Reserved	Reserved	Reserved
3	11	0.01°F	Reserved	Reserved	Reserved

IL_INLN_TEMP_DATA The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value in corresponding resolution
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 598

Tab. 14-55: Structural elements IL_INLN_TEMP_DATA

IL_INLN_DIAG_CODE The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-56: Enum IL_INLN_DIAG_CODE

IL_INLN_TEMP_R0 The resistance reference value to configure the channel can be selected using this enumeration type.

Enum	Description
R10	Resistance reference value R0: 10 Ω
R20	Resistance reference value R0: 20 Ω
R30	Resistance reference value R0: 30 Ω
R50	Resistance reference value R0: 50 Ω
R100	Resistance reference value R0: 100 Ω
R120	Resistance reference value R0: 120 Ω
R150	Resistance reference value R0: 150 Ω
R200	Resistance reference value R0: 200 Ω
R240	Resistance reference value R0: 240 Ω
R300	Resistance reference value R0: 300 Ω
R400	Resistance reference value R0: 400 Ω
R500	Resistance reference value R0: 500 Ω

Enum	Description
R1000	Resistor reference value R0: 1000 Ω
R1500	Resistor reference value R0: 1500 Ω
R2000	Resistance reference value R0: 2000 Ω
R10000	Resistance reference value R0: 10000 Ω
R3000	Only for the IL_Temp2Rtd function block

Tab. 14-57: Enum IL_INLN_TEMP_R0 for IL_AI8Sfls

IL_INLN_TEMP_SENSOR

The sensor type to configure a channel can be selected using this enumeration type.

Enum	Sensor type
PTDIN	PT DIN
PTSAMA	PT SAMA
NIDIN	NI DIN
NISAMA	NI SAMA
CU10	CU10
CU50	CU50
CU53	CU53
NI1000	NI1000 (Landis & Gyr)
NI500	NI500 (Viesmann)
KTY81110	KTY81-100
KTY84	KTY84
KTY81210	KTY81210
POT	POTENTIOMETER (%)
LINEAR400	Linear R: 0 ... 400 Ω
LINEAR20000	Linear R: 0 ... 20000 Ω

Tab. 14-58: Enum IL_INLN_TEMP_SENSOR for the IL_Temp4_8Rtd function block

IL_INLN_TEMP_CONNECTION-TYPE

The connection type of the sensor can be selected using this enumeration type.

Enum	Description
THREEWIRE	Connection type: 3-wire sensor
TWOWIRE	Connection type: 2-wire sensor
FOURWIRE	Reserved for the IL_Temp2Rtd function block

Tab. 14-59: Enum IL_INLN_TEMP_CONNECTIONTYPE for IL_Temp4_8Rtd

Example

The function block configures an RTD temperature terminal for the temperature measurement.

Make the following configuration presettings: Configuration of inputs, channel selection and process data mapping to the device.

In this example, Temp4_8Rtd is configured as follows: Channels 1 and 2 for the sensors "PTDIN" and "POT".

To read data, the "NormalizedValue" of the output structure is available for each channel.

Declaration of the example

```
PROGRAM PrgIL_Temp4_8Rtd
VAR
    fbIL_Temp4_8Rtd: IL_Temp4_8Rtd;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wInProcessData_2: WORD;
    wInProcessData_3: WORD;
    wInProcessData_4: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    rNormValCH1: REAL;
    rNormValCH2: REAL;
    enDiagCodeCH1: IL_INLN_DIAG_CODE;
    enDiagCodeCH2: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_Temp4_8Rtd.ChannelConfig[0].ConnectionType := IL_INLN_TEMP_CONNECTIONTYPE.THREEWIRE;
 fbIL_Temp4_8Rtd.ChannelConfig[0].NoOfSamples    := BYTE#16;
 fbIL_Temp4_8Rtd.ChannelConfig[0].Resistance     := IL_INLN_TEMP_R0.R100;
 fbIL_Temp4_8Rtd.ChannelConfig[0].Resolution     := WORD#1;
 fbIL_Temp4_8Rtd.ChannelConfig[0].SensorType     := IL_INLN_TEMP_SENSOR.PTDIN;
 fbIL_Temp4_8Rtd.ChannelConfig[1].ConnectionType := IL_INLN_TEMP_CONNECTIONTYPE.TWOWIRE;
 fbIL_Temp4_8Rtd.ChannelConfig[1].NoOfSamples    := BYTE#32;
 fbIL_Temp4_8Rtd.ChannelConfig[1].Resistance     := IL_INLN_TEMP_R0.R1000;
 fbIL_Temp4_8Rtd.ChannelConfig[1].Resolution     := WORD#0;
 fbIL_Temp4_8Rtd.ChannelConfig[1].SensorType     := IL_INLN_TEMP_SENSOR.POT;
 fbIL_Temp4_8Rtd.ChannelSelect                   := BYTE#2#00000011;
 fbIL_Temp4_8Rtd.InProcData_0         := wInProcessData_0;
 fbIL_Temp4_8Rtd.InProcData_1         := wInProcessData_1;
 fbIL_Temp4_8Rtd.InProcData_2         := wInProcessData_2;
 fbIL_Temp4_8Rtd.InProcData_3         := wInProcessData_3;
 fbIL_Temp4_8Rtd.InProcData_4         := wInProcessData_4;
 wOutProcessData_0   := fbIL_Temp4_8Rtd.OutProcData_0;
 wOutProcessData_1   := fbIL_Temp4_8Rtd.OutProcData_1;

 // Call
 fbIL_Temp4_8Rtd(Enable:= TRUE );

 // Data read
 IF fbIL_Temp4_8Rtd.InOperation = TRUE THEN
  rNormValCH1    := fbIL_Temp4_8Rtd.OutputData[0].NormalizedValue;
  rNormValCH2    := fbIL_Temp4_8Rtd.OutputData[1].NormalizedValue;
  enDiagCodeCH1  := fbIL_Temp4_8Rtd.OutputData[0].DiagCode;
  enDiagCodeCH2  := fbIL_Temp4_8Rtd.OutputData[1].DiagCode;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_Temp4_8Rtd".

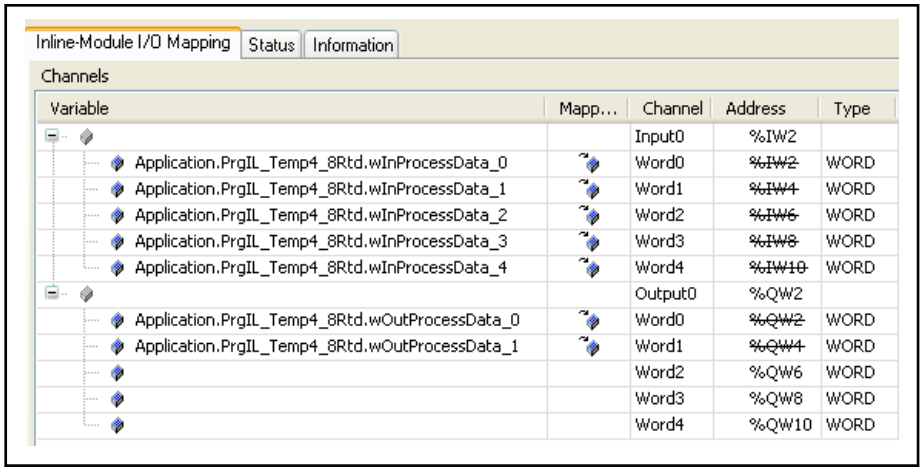


Fig. 14-17: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003005	Channel number	The "NoOfSamples" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003006	Channel number	“Resistance” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003007	Channel number	The "Resolution" input of a channel is invalid or does not match the resolution of the sensor type
INPUT_INVALID_ERROR (16#0001)	16#00003008	Channel number	"SensorType" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003014	Channel number	“ConnectionType" of a channel is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#00003020	16#000x8010	Invalid configuration

Tab. 14-60: Error codes of the IL_Temp4_8Rtd

14.2.10 IL_Temp4_8RtdEf

Brief description

The **IL_Temp4_8RtdEf** function block initializes and configures the Inline terminal **R-IB IL TEMP 4/8 RTD-EF-PAC** with eight resistive temperature sensor inputs and normalizes data according to the user-defined nominal resolution. **R-IB IL TEMP 4/8 RTD-EF-PAC** supports additional 4-wire sensors compared to the **R-IB IL TEMP 4/8 RTD-PAC**.

Compatibility

The function block can be used for Inline bus, Profibus and Profinet IO field bus.

Interface description

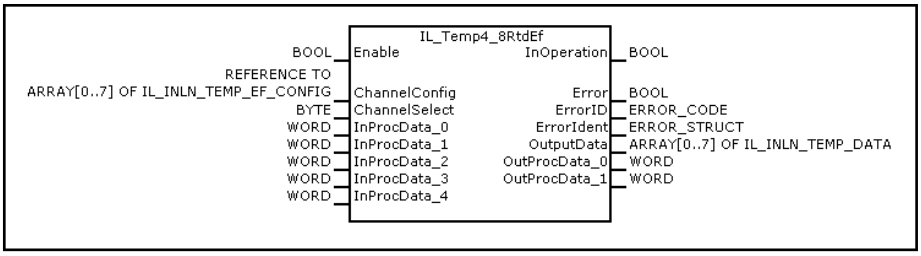


Fig. 14-18: IL_Temp4_8RtdEf function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	ChannelConfig	REFERENCE TO ARRAY[0..7] OF IL_INLN_TEMP_EF_CONFIG	Contains configuration data for the channels 1...8, see "IL_INLN_TEMP_EF_CONFIG" on page 603
	ChannelSelect	BYTE	Byte#2#11111111; Individual channel activation (bit 0... bit 7 correspond to channel 1...8)
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
	InProcData_2	WORD	Process input word 2
	InProcData_3	WORD	Process input word 3
	InProcData_4	WORD	Process input word 4
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 607
	OutputData	ARRAY[0..7] OF IL_INLN_TEMP_DATA	Contains the normalized value and the diagnostic code for each channel, see "IL_INLN_TEMP_DATA" on page 604
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-61: Interface IL_Temp4_8RtdEf

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ChannelSelect	BYTE	1	255	255	Positive edge at "Enable"

Tab. 14-62: Input behavior of the IL_Temp4_8RtdEf

Functional Description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

To parameterize the channel, the necessary values have to be assigned to the structure **IL_INLN_TEMP_EF_CONFIG**. The channel is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

The initial values of the function block are adjusted with the initial values of the analog terminal.

To obtain the measured values or diagnostic information of the channels 1 ... 8 of the terminal, set **ChannelSelect** accordingly.

The channels can be read with the enabled bits of **ChannelSelect**. Bit 0 corresponds to channel 1 and up to bit 7 to channel 8.

All channels are selected by default.

The measured values are normalized according to the user-defined resolution and displayed in **NormalizedValue**. If an error occurs, it is displayed for the respective channel in **DiagCode**.

☞ R-IB IL TEMP4/8-RTD-EF-PAC operates in multiplex (data is thus also read in multiplex). Updating data for each cycle thus requires longer than one PLC cycle.

IL_INLN_TEMP_EF_CONFIG

The structure contains information on one channel configuration. The channel is defined by **SensorType**, **Resolution**, **Resistance**, **FilterTime** and **ODSSensitivty**.

It is configured as follows:

```
ParameterName[Channel_Number].[Function].
```

e.g. to configure **SensorType** when calling channel 1 for the first time:

```
ChannelConfig[0].SensorType.
```

Name	Type	Description	Min. value	Max. value	Default value	Effective
SensorType	IL_INLN_TEMP_EF_SENSOR	Sensor type selection, see "IL_INLN_TEMP_EF_SENSOR" on page 605 Example: PTSAMA	PTDIN	LINEAR5000	PTDIN	Positive edge at "Enable"
Resolution	BYTE	Resolution: **0 : 0.1°C/1%/0.1 Ω** 1 : 0.01°C/0.1%/0.01Ω 2 : 0.1°F/ reserved 3 : 0.01°F/ reserved Refer to the resolution table below	0	3	0	Positive edge at "Enable"
Resistance	IL_INLN_TEMP_EF_R0	Resistance reference value R0 [Ω], see "IL_INLN_TEMP_EF_R0" on page 605	R100	R10000	R100	Positive edge at "Enable"

Name	Type	Description	Min. value	Max. value	Default value	Effective
FilterTime	BYTE	0 - 480 ms 1 - 120 ms 2 - 101ms 3 - 200ms	0	3	0	
ODSSensitivity	IL_INLN_TEMP_EF_ODS	Sensitivity of the wire break detection "IL_INLN_TEMP_EF_ODS" on page 605	0	3	0	Positive edge at "Enable"

Tab. 14-63: Permitted values of IL_INLN_TEMP_EF_CONFIG for the IL_Temp4_8RtdEf function block

Resolution table

Code		Resolution for sensor type			
Dec.	Bin.	0 ...11	Linear R0...500 Ω	Linear R0...5 kΩ	Linear R0...30 kΩ
0	00	0.1°C	0.1 Ω	1 Ω	1 Ω
1	01	0.01°C	0.01 Ω	0.1 Ω	Reserved
2	10	0.1°F	Reserved	Reserved	Reserved
3	11	0.01°F	Reserved	Reserved	Reserved

IL_INLN_TEMP_DATA

The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value in corresponding resolution
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code, see "IL_INLN_DIAG_CODE" on page 604

Tab. 14-64: Structural elements IL_INLN_TEMP_DATA

IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty

DiagCode	ENUM	Description
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-65: Enum IL_INLN_DIAG_CODE

IL_INLN_TEMP_EF_R0 The resistance reference value to configure the channel can be selected using this enumeration type.

```
TYPE IL_INLN_TEMP_EF_R0 :
(
    R100   := 0,    // Resistance of Sensor at 0deg  : 100 ohm
    R10    := 1,    // Resistance of Sensor at 0deg  : 10 ohm
    R20    := 2,    // Resistance of Sensor at 0deg  : 20 ohm
    R30    := 3,    // Resistance of Sensor at 0deg  : 30 ohm
    R50    := 4,    // Resistance of Sensor at 0deg  : 50 ohm
    R120   := 5,    // Resistance of Sensor at 0deg  : 120 ohm
    R150   := 6,    // Resistance of Sensor at 0deg  : 150 ohm
    R200   := 7,    // Resistance of Sensor at 0deg  : 200 ohm
    R240   := 8,    // Resistance of Sensor at 0deg  : 240 ohm
    R300   := 9,    // Resistance of Sensor at 0deg  : 300 ohm
    R400   := 10,   // Resistance of Sensor at 0deg  : 400 ohm
    R500   := 11,   // Resistance of Sensor at 0deg  : 500 ohm
    R1000  := 12,   // Resistance of Sensor at 0deg  : 1000 ohm
    R1500  := 13,   // Resistance of Sensor at 0deg  : 1500 ohm
    R2000  := 14,   // Resistance of Sensor at 0deg  : 2000 ohm
    R10000 := 15,   // Resistance of Sensor at 0deg  : 10000 ohm
    R3000  := 16    // Resistance of Sensor at 0deg  : 3000 ohm
) WORD;
END_TYPE
```

IL_INLN_TEMP_EF_SENSOR The sensor type to configure a channel can be selected using this enumeration type.

```
TYPE IL_INLN_TEMP_EF_SENSOR :
(
    PTDIN       := 0,    //  Pt DIN SENSOR TYPE
    PTSAMA      := 1,    //  Pt SAMA SENSOR TYPE
    NIDIN       := 2,    //  Ni DIN SENSOR TYPE
    NISAMA      := 3,    //  Ni SAMA SENSOR TYPE
    CU10        := 4,    //  Cu 10 SENSOR TYPE
    CU50        := 5,    //  Cu 50 SENSOR TYPE
    CU53        := 6,    //  Cu 53 SENSOR TYPE
    NI1000      := 7,    //  Ni 1000 (LANDIS & GYR) SENSOR TYPE
    NI500       := 8,    //  Ni 500 (VIESSMANN) SENSOR TYPE
    KTY81110    := 9,    //  KTY81 -110 SENSOR TYPE
    KTY84       := 16#A, //  KTY84 SENSOR TYPE
    KTY81210    := 16#B, //  KTY81-210 SENSOR TYPE
    LINEAR30000 := 16#C, //  LINEAR R : 0 TO 30000 Ohm SENSOR TYPE

    LINEAR500   := 16#E, //  LINEAR R :0 TO 500 Ohm SENSOR TYPE
    LINEAR5000  := 16#F, //  LINEAR R :0 TO 5000 Ohm SENSOR TYPE
 ) WORD;
END_TYPE
```

IL_INLN_TEMP_EF_ODS This enumeration type allows to select the sensitivity of the wire break detection. (ODS - **o**pen circuit **d**etection **s**ensitivity): "High sensitivity"/ "Average sensitivity"/ "Off".

Type definition

```
TYPE IL_INLN_TEMP_EF_ODS:
(
HIGH_SENSITIVITY := 0   // High Sensitivity
MEDIUM_SENSITIVITY := 1 // Medium Sensitivity
SWITCHED_OFF := 3       // Switched Off
) BYTE;
END_TYPE
```

Example The function block configures an RTD EF temperature terminal for the temperature measurement.

Make the following configuration presettings: Configuration of inputs, channel selection and process data mapping to the device.

In this example, Temp4_8RtdEf is configured as follows: Channels 1 and 2 for the sensors "PTDIN" and "Linear Resistance 5000 Ohms".

To read data, the "NormalizedValue" of the output structure is available for each channel.

Declaration of the example

```
PROGRAM PrgIL_Temp4_8RtdEf
VAR
    fbIL_Temp4_8RtdEf: IL_Temp4_8RtdEf;
 bEnable: BOOL;
 byChannelSelect: BYTE := 255;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wInProcessData_2: WORD;
    wInProcessData_3: WORD;
    wInProcessData_4: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    rNormValCH1: REAL;
    rNormValCH2: REAL;
    enDiagCodeCH1: IL_INLN_DIAG_CODE;
    enDiagCodeCH2: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration Channel 1
 fbIL_Temp4_8RtdEf.ChannelConfig[0].SensorType    := IL_INLN_TEMP_EF_SENSOR.PTDIN;
 fbIL_Temp4_8RtdEf.ChannelConfig[0].Resolution    := BYTE#0;
 fbIL_Temp4_8RtdEf.ChannelConfig[0].Resistance    := IL_INLN_TEMP_EF_R0.R100;
 fbIL_Temp4_8RtdEf.ChannelConfig[0].FilterTime    := BYTE#0;
 fbIL_Temp4_8RtdEf.ChannelConfig[0].ODSSensitivity:= IL_INLN_TEMP_EF_ODS.HIGH_SENSITIVITY;
// Configuration Channel 2
 fbIL_Temp4_8RtdEf.ChannelConfig[1].SensorType    := IL_INLN_TEMP_EF_SENSOR.LINEAR5000;
 fbIL_Temp4_8RtdEf.ChannelConfig[1].Resolution    := BYTE#0;
 fbIL_Temp4_8RtdEf.ChannelConfig[1].Resistance    := IL_INLN_TEMP_EF_R0.R2000;
 fbIL_Temp4_8RtdEf.ChannelConfig[1].FilterTime    := BYTE#0;
 fbIL_Temp4_8RtdEf.ChannelConfig[1].ODSSensitivity:= IL_INLN_TEMP_EF_ODS.SWITCHED_OFF;

 byChannelSelect := BYTE#2#00000011; // Channel 1 & 2 selected

 fbIL_Temp4_8Rtd.InProcData_0         := wInProcessData_0;
 fbIL_Temp4_8Rtd.InProcData_1         := wInProcessData_1;
 fbIL_Temp4_8Rtd.InProcData_2         := wInProcessData_2;
 fbIL_Temp4_8Rtd.InProcData_3         := wInProcessData_3;
 fbIL_Temp4_8Rtd.InProcData_4         := wInProcessData_4;

 wOutProcessData_0   := fbIL_Temp4_8RtdEf.OutProcData_0;
 wOutProcessData_1   := fbIL_Temp4_8RtdEf.OutProcData_1;

 // FB call
 fbIL_Temp4_8RtdEf(Enable:= TRUE );

 // Data read
 IF fbIL_Temp4_8RtdEf.InOperation = TRUE THEN
  rNormValCH1    := fbIL_Temp4_8RtdEf.OutputData[0].NormalizedValue;
  rNormValCH2    := fbIL_Temp4_8RtdEf.OutputData[1].NormalizedValue;
  enDiagCodeCH1  := fbIL_Temp4_8RtdEf.OutputData[0].DiagCode;
  enDiagCodeCH2  := fbIL_Temp4_8RtdEf.OutputData[1].DiagCode;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_Temp4_8RtdEf".

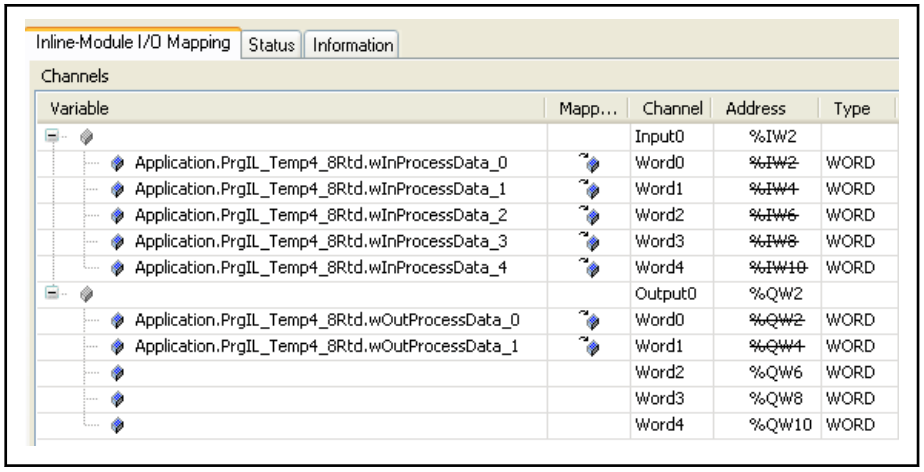


Fig. 14-19: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	Invalid “ChannelSelect” input
INPUT_INVALID_ERROR (16#0001)	16#00003005	Channel number	“NoOfSamples/FilterTime" of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003006	Channel number	“Resistance” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003007	Channel number	The "Resolution" input of a channel is invalid or does not match the resolution of the sensor type
INPUT_INVALID_ERROR (16#0001)	16#00003008	Channel number	"SensorType" input of a channel is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003019	Channel number	“ODSSensitivity" of a channel is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#00003020	16#000x8010	Invalid configuration

Tab. 14-66: Error codes of the IL_Temp4_8RtdEf

14.3 Analog output terminals

14.3.1 IL_AO1Sf

Brief description

The function block **ILAO1Sf** converts a normalized value and applies it to the analog output of the **R-IB IL AO-1/SF** terminal (Inline terminal with an analog output).

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos and Profinet IO field bus.

Interface description

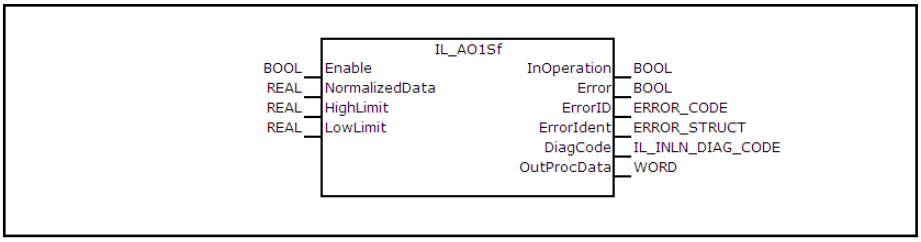


Fig. 14-20: IL_AO1Sf function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	NormalizedValue	REAL	Normalized input value
	HighLimit	REAL	Upper limit for normalized input value
	LowLimit	REAL	Lower limit for normalized input value
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 609
	DiagCode	IL_INLN_DIAG_CODE	Diagnostic information, see "IL_INLN_DIAG_CODE" on page 608
	OutProcData	WORD	Process output word

Tab. 14-67: IL_AO1Sf interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
NormalizedValue	REAL	-65535.0	65535.0	0.0	Continuously with "Enable"
HighLimit	REAL	-65535.0	65535.0	10.0	Continuously with "Enable"
LowLimit	REAL	-65535.0	65535.0	0.0	Continuously with "Enable"

Tab. 14-68: Input behavior of the IL_AO1Sf

Functional description

The **IL_AO1Sf** function block cyclically converts the normalized input value into the analog output value of the **R-IB-IL-AO-1/SF** module. The **R-IB-IL-AO-1/SF** module is provided with an analog output that can either connect voltage or current signals (0-10 V or 0-20 mA / 4-20 mA). The different areas can be used by the wiring of different connection points at the module. For the wiring, refer to the hardware documentation of the module.

IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-69: Enum IL_INLN_DIAG_CODE

Example

In this example, the function block configures the **R-IB-IL-AO-1/SF** module for voltage outputs between 0 V and 10 V.

Declaration of the example

```
PROGRAM PrgIL_AO1Sf
VAR
    fbIL_AO1Sf: IL_AO1Sf;
    wOutProcessData: WORD;
    enDiagCode: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration
fbIL_AO1Sf.HighLimit := REAL#10.0 ;
fbIL_AO1Sf.LowLimit := REAL#0.0 ;
wOutProcessData := fbIL_AO1Sf.OutProcData;

 // Call
fbIL_AO1Sf(    Enable:=TRUE);

 // Data output
IF    fbIL_AO1Sf.InOperation = TRUE THEN
    fbIL_AO1Sf.NormalizedValue := REAL#5.0 ;
    enDiagCode := fbIL_AO1Sf.DiagCode ;
END_IF
```

Process data interface of the example

Map the process output word of the function block on the process data word of the individual field bus system.

The "wOutProcessData" variable is declared in the "prgIL_AO1Sf" program.

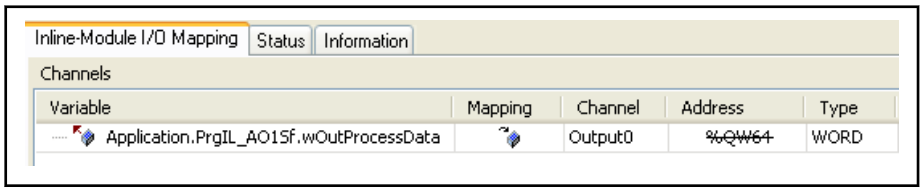


Fig. 14-21: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003009	16#00000000	Invalid range limits HighLimit <= LowLimit

Tab. 14-70: Error codes of the IL_AO1Sf

14.3.2 IL_AO2SfBp

Brief description

The **IL_AO2SfBp** function block parameterizes the Inline terminals **R IB-IL-AO-2/SF** (Inline terminal with two analog outputs) and **R IB-IL-AO-2/U/BP** and converts and applies normalized values to their analog outputs.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus and Profinet IO field bus.

Interface description

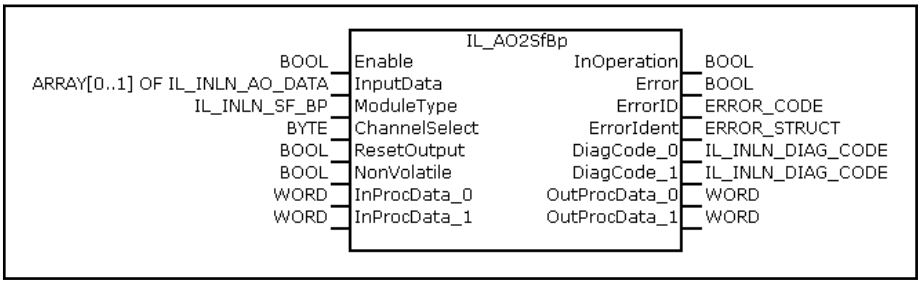


Fig. 14-22: IL_AO2SfBp function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	InputData	ARRAY[0..1] OF IL_INLN_AO_DATA	Data transfer to channels "IL_INLN_AO_DATA" on page 611
	ModuleType	IL_INLN_SF_BP	Module type selection: Example: SF, "IL_INLN_SF_BP" on page 612
	ChannelSelect	BYTE	Channel selection, BYTE#2#000000yx; x - channel 1, y - channel 2
	ResetOutput	BOOL	Output behavior: FALSE – HOLT, TRUE – RESET
	NonVolatile	BOOL	Output behavior: FALSE – volatile, TRUE – non-volatile
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information "Troubleshooting" on page 613
	DiagCode_0	IL_INLN_DIAG_CODE	Diagnostic information on channel 0 "IL_INLN_DIAG_CODE" on page 612
	DiagCode_1	IL_INLN_DIAG_CODE	Diagnostic information on channel 1
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-71: IL_AO2SfBp interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
ModuleType	ENUM	SF	BPBIPOLAR10V	SF	Positive edge at "Enable"
ChannelSelect	BYTE	0	3	3	Continuously with "Enable"
ResetOutput	BOOL	FALSE	TRUE	FALSE	Positive edge at "Enable"
NonVolatile	BOOL	FALSE	TRUE	FALSE	Positive edge at "Enable"

Tab. 14-72: Input behavior of the IL_AO2SfBp

Functional description

The function block **IL_AO2SfBp** parameterizes an **R IB-IL-AO-2/SF** or an **R IB-IL-AO-2/U/BP** terminal depending on the selected **ModuleType**. It is parameterized via a positive edge at the **Enable** input. The function block converts the normalized values and applies them to the analog outputs of the channels 1 and/or 2. If an error occurs, the function block output **DiagCode** and the error outputs are provided with the corresponding error information.

The output behavior in error-free operation (normal operation): In normal operation mode, the parameterized output range and the data format are permanently read when adding the voltage of the terminal (power up). If an **error** occurs, there can also be a volatile parameterization of these settings and the behavior of the terminal. This can still be parameterized at runtime using a process data sequence.

Output behavior if an error occurs: If an error occurs, the outputs respond as set for permanent parameterization or subsequent volatile parameterization. Thus, the outputs keep the last value (HOLD, default setting) or return to zero (RESET).

IL_INLN_AO_DATA

The structure describes the input data of one channel.

Name	Type	Description	Min. value	Max. value	Default value	Effective
NormalizedValue	REAL	Normalized input value for the analog output	LowLimit	HighLimit	0.0	Continuously with "Enable"
HighLimit	REAL	Upper limit of the normalized input value	-	-	10.0	Continuously with "Enable"
LowLimit	REAL	Lower limit of the normalized input value	-	-	0.0	Continuously with "Enable"

IL_INLN_SF_BP This enumeration selected the type of the analog output terminal by the function block.

ENUM	Description
SF	**R-IB IL AO 2/SF** module with the output range **0..10 V, 0..20 mA** and **4..20 mA**
BPUNIPOLAR10V	**R-IB IL AO 2/U/BP** module with the output range **0 ..10 V**
BPBIPOLAR10V	**R-IB IL AO 2/U/BP** module with the output range **+10 ..-10 V**

IL_INLN_DIAG_CODE The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-73: Enum IL_INLN_DIAG_CODE

Example The function block configures the terminals **R IB-IL AO2/Sf** and **R IB-IL AO2/U/BP**. In the **R IB-IL AO2/Sf** module, make the selection 0...10 V, 0...20 mA and 4 mA... 20 mA using hardware bridges.

In the following example, the function block configures an **R IB-IL AO2/Sf** module in channel 1 to 0...10 V and in channel 2 to 4 .. 20 mA. However, channel 2 has to be provided with correct hardware bridges.

Specify the connected module type via the **ModuleType** variable. The upper and lower limit for channel 1 are 10 and 0. Thus, the output 0...10 V follows the normalized value between 0 and 10.

Declaration of the example

```
PROGRAM PrgIL_AO2SfBp
VAR
    fbIL_AO2SfBp: IL_AO2SfBp;
```

```
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: INT;
    wOutProcessData_1: WORD;
    enDiagCode_0: IL_INLN_DIAG_CODE;
    enDiagCode_1: IL_INLN_DIAG_CODE;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_AO2SfBp.ModuleType := IL_INLN_SF_BP.SF ;
 fbIL_AO2SfBp.ResetOutput := TRUE ;
 fbIL_AO2SfBp.NonVolatile := TRUE ;
 fbIL_AO2SfBp.InputData[0].HighLimit := REAL#10.0 ;
 fbIL_AO2SfBp.InputData[0].LowLimit := REAL#0.0 ;
 fbIL_AO2SfBp.InputData[1].HighLimit := REAL#20.0 ;
 fbIL_AO2SfBp.InputData[1].LowLimit := REAL#4.0 ;
 fbIL_AO2SfBp.ChannelSelect := BYTE#2#00000011 ;
 fbIL_AO2SfBp.InProcData_0 := wInProcessData_0 ;
 fbIL_AO2SfBp.InProcData_1 := wInProcessData_1 ;
 wOutProcessData_0 := fbIL_AO2SfBp.OutProcData_0 ;
 wOutProcessData_1 := fbIL_AO2SfBp.OutProcData_1 ;

 // FB call
 fbIL_AO2SfBp(Enable:= TRUE);

 // Data output
  (* Once the function block reaches the in operation mode the module starts
     generating voltage and current signals.
     The out can be varying by varying the value at Normalized value input.*)
 IF    fbIL_AO2SfBp.InOperation = TRUE THEN
     fbIL_AO2SfBp.InputData[0].NormalizedValue := REAL#10.0 ;
     fbIL_AO2SfBp.InputData[1].NormalizedValue := REAL#15.0 ;
     enDiagCode_0 := fbIL_AO2SfBp.DiagCode_0 ;
     enDiagCode_1 := fbIL_AO2SfBp.DiagCode_1 ;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "arInProcessData_x" and "arOutProcessData_x" are declared in the program "prgIL_AO2SfBp".

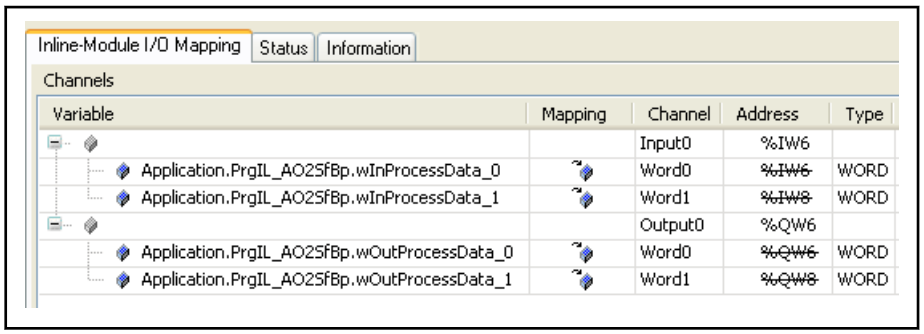


Fig. 14-23: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	ChannelSelect is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003009	channel number	The limit values are incorrect HighLimit <= LowLimit.
INPUT_INVALID_ERROR (16#0001)	16#00003010	16#00000000	ModuleType is invalid

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Timeout, the device is not available. Check the module type
DEVICE_ERROR (16#0008)	16#00003020	16#00008010	Invalid configuration or contradictive hardware bridges

Tab. 14-74: Error codes of the IL_AO2SfBp

14.3.3 IL_AO2UI

Brief description

The **IL_AO2UI** function block supports the configuration of the two-channel Inline analog output terminal **R-IB IL AO 2/UI-PAC**.

After a successful configuration, the function block converts normalized input values per channel and applies them to the analog outputs of the Inline terminal.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus and Profinet IO field bus.

Interface description

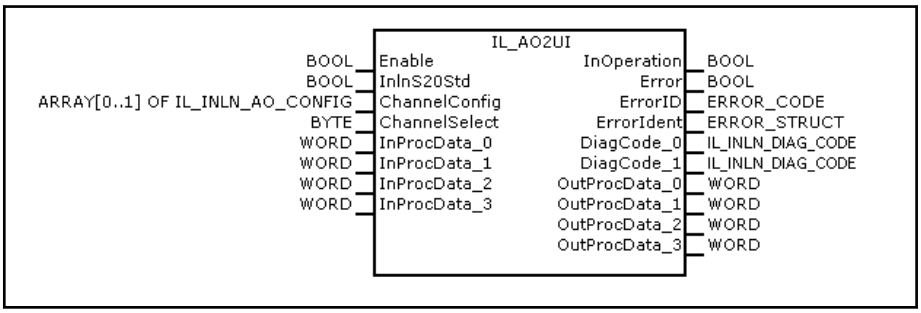


Fig. 14-24: IL_AO2UI function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	InInS20Std	BOOL	Module type selection: TRUE: S20 FALSE. Inline
	ChannelConfig	ARRAY[0..1] OF IL_INLN_AO_CONFIG	Analog output channel configuration IL_INLN_AO_CONFIG , page 615
	ChannelSelect	BYTE	Channel selection, BYTE#2#000000yx; x - channel 1, y - channel 2
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
	InProcData_2	WORD	Process input word 2
	InProcData_3	WORD	Process input word 3
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID

	Name	Type	Description
	ErrorIdent	ERROR_STRUCT	Detailed error information Troubleshooting, page 618
	DiagCode_0	IL_INLN_DI-AG_CODE	Diagnostic information on channel 0 IL_INLN_DIAG_CODE, page 616
	DiagCode_1	IL_INLN_DI-AG_CODE	Diagnostic information on channel 1
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1
	OutProcData_2	WORD	Process output word 2
	OutProcData_3	WORD	Process output word 3

Tab. 14-75: IL_AO2UI interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
InInS20Std	BOOL	FALSE	TRUE	FALSE	Positive edge at "Enable"
ChannelSelect	BYTE	0	3	3	Continuously with "Enable"

Tab. 14-76: Input behavior of the IL_AO2UI

Functional description

The IL_AO2UI function block cyclically converts normalized input values into analog values for the R-IB IL AO 2/UI-PAC terminal. The R-IB IL AO 2/UI-PAC terminal consists of two analog output channels with two voltage ranges: 0 V ... 10 V, -10 V ... +10 V or the current ranges: 0 mA ... 20 mA, 4 mA ... 20 mA, ±20 mA.

Configure the output ranges via the enumeration type IL_INLN_AO_RANGE, "ChannelConfig.OuputRange".

The "InInS20Std" input decides whether to use INLINE support (FALSE) or S20 support (TRUE).

The channels are configured with a positive edge at "Enable".

After completion, the "InOperation" output is set to "TRUE".

It can only be configured again if there is a new positive edge at "Enable".

If the configuration is invalid, the function block generates an error: "Error", "ErrorID" and "ErrorIdent" are set accordingly.

The initial values of the function block parameterization correspond to the initial values of the analog terminal.

The function block converts normalized input values into analog values for channel 1 and/or 2 and outputs them.

If an error occurs in one of the channels, the error variable is set to TRUE and the "DiagCode" indicates error information on the specified channel.

IL_INLN_AO_CONFIG

The structure describes the input data of **one** channel.

Name	Type	Description	Min. value	Max. value	Default value	Effective
NormalizedValue	REAL	Normalized input value for the analog output	LowLimit	HighLimit	0.0	Continuously with "Enable"
HighLimit	REAL	Upper limit of the normalized input value	-	-	10.0	Positive edge at "Enable"
LowLimit	REAL	Lower limit of the normalized input value	-	-	0.0	Positive edge at "Enable"
OutputRange	IL_INLN_AO_RANGE	Measuring range of analog output voltage/current	UNIPOLAR10V	UNIPOLAR4TO20MA	UNIPOLAR10V	Positive edge at "Enable"

IL_INLN_AO_RANGE This enumeration is used to select a suitable output channel voltage or a suitable output channel current by the function block.

ENUM	Value	Description
UNIPOLAR10V	0	Measuring voltage from 0 to 10 V
BIPOLAR10V	1	Measuring voltage from -10 to +10 V
UNIPOLAR20MA	8	Measuring voltage from 0 to 20 mA
BIPOLAR20MA	9	Measuring current from -20 to +20 mA
UNIPOLAR4TO20MA	10	Measuring current from 4 to 20 mA

IL_INLN_DIAG_CODE The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-77: Enum IL_INLN_DIAG_CODE

Example The function block configures the **R IB-IL AO 2/UI** module for channel 1 to 0...10 V "arChannelConfig[0].OutputRange" and for channel 2 to +/- 10 V "arChannelConfig[1].OutputRange".

The normalized input data "arChannelConfig[x].NormalizedValue" is between "arChannelConfig[x].HighLimit" and "arChannelConfig[x].LowLimit" and results in the output data at the process output data of the corresponding channel.

Declaration of the example

```
PROGRAM prgIL_AO2UI
VAR
```

```
    fbIL_AO2UI:        IL_AO2UI;
    bExecute:        BOOL:= FALSE;
    bInlnS20Std:         BOOL:= FALSE;  // FALSE: Inline Standard, TRUE:S20 Standard
    arChannelConfig:    ARRAY [0..1] OF IL_INLN_AO_CONFIG; // Analog Channel Configuration
    byChannelSelect:    BYTE := 3;     // Channel Selection
    wInProcData_0:    WORD;        // Input Process Data Word 1
    wInProcData_1:    WORD;        // Input Process Data Word 2
    wInProcData_2:    WORD;        // Input Process Data Word 3
    wInProcData_3:    WORD;        // Input Process Data Word 4
    bInOperation:        BOOL;
    bError:            BOOL;
    enErrorID:        ERROR_CODE;
    enErrorIdent:        ERROR_STRUCT;
    enDiagCode_0:        IL_INLN_DIAG_CODE; // Diagnostic code Channel -1
    enDiagCode_1:        IL_INLN_DIAG_CODE; // Diagnostic code Channel -2
    wOutProcData_0:    WORD;         // Output Process Data Word 1
    wOutProcData_1:    WORD;             // Output Process Data Word 2
    wOutProcData_2:    WORD;             // Output Process Data Word 3
    wOutProcData_3:    WORD;             // Output Process Data Word 4

    rNormalizedValue_0:     REAL     :=  5.0;      //Normalized Value for Channel 1
    rNormalizedValue_1:     REAL    := -5.0;        //Normalized Value for Channel 2
END_VAR
```

Implementation of the example

```
//Channel 1 Configuration : UNIPOLAR10V 0 to 10V
arChannelConfig[0].LowLimit        := REAL#0.0;
arChannelConfig[0].HighLimit        := REAL#10.0;
arChannelConfig[0].OutputRange      := IL_INLN_AO_RANGE.UNIPOLAR10V;

//Channel 2 Configruation : BIPOLAR10V -10V to +10V
arChannelConfig[1].LowLimit        := REAL#-10.0;
arChannelConfig[1].HighLimit        := REAL#10.0;
arChannelConfig[1].OutputRange      := IL_INLN_AO_RANGE.BIPOLAR10V;

//FB Call
fbIL_AO2UI(
    Enable:=bExecute ,
    InlnS20Std:=bInlnS20Std ,
    ChannelConfig:= arChannelConfig ,
    ChannelSelect:= byChannelSelect ,
    InProcData_0:= wInProcData_0 ,
    InProcData_1:= wInProcData_1 ,
    InProcData_2:= wInProcData_2 ,
    InProcData_3:= wInProcData_3,
    InOperation=> bInOperation,
    Error=> bError ,
    ErrorID=> enErrorID ,
    ErrorIdent=> enErrorIdent ,
    DiagCode_0=> enDiagCode_0 ,
    DiagCode_1=> enDiagCode_1,
    OutProcData_0=> wOutProcData_0 ,
    OutProcData_1=> wOutProcData_1 ,
    OutProcData_2=> wOutProcData_2,
    OutProcData_3=> wOutProcData_3);

Data Write:

//Passing Normalized Values to channels
IF fbIL_AO2UI.InOperation = TRUE THEN
    arChannelConfig[0].NormalizedValue := rNormalizedValue_0;
    arChannelConfig[1].NormalizedValue := rNormalizedValue_1;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "arInProcessData_x" and "arOutProcessData_x" are declared in the program "prgIL_AO2UI".

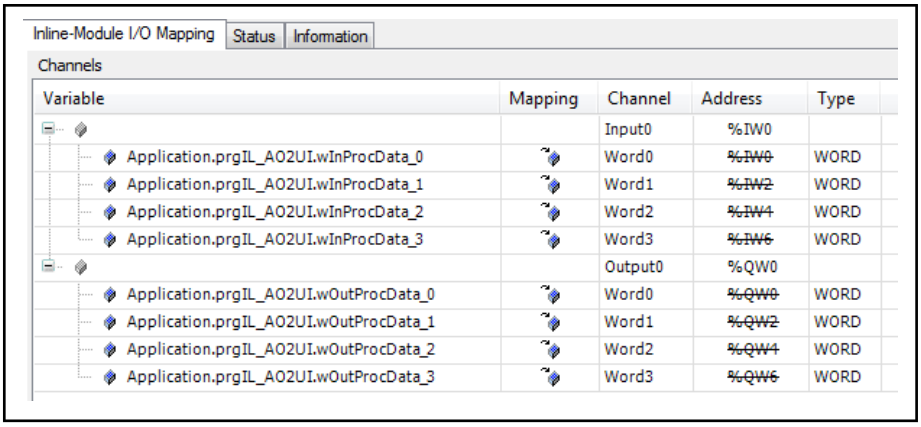


Fig. 14-25: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003003	16#00000000	ChannelSelect is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003004	channel number	OutputRange is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003009	channel number	The limit values are incorrect HighLimit <= LowLimit.
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Timeout, the device is not available. Check the module type
DEVICE_ERROR (16#0008)	16#00003020	16#00008010	Invalid module configuration

Tab. 14-78: Error codes of the IL_AO2UI

14.4 Function terminals

14.4.1 IL_CntCount

Brief description

The **IL_CntCount** function block parameterizes the **R-IB IL CNT** terminal (Inline counter terminal) in the "event counting" mode and reads the counting result.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

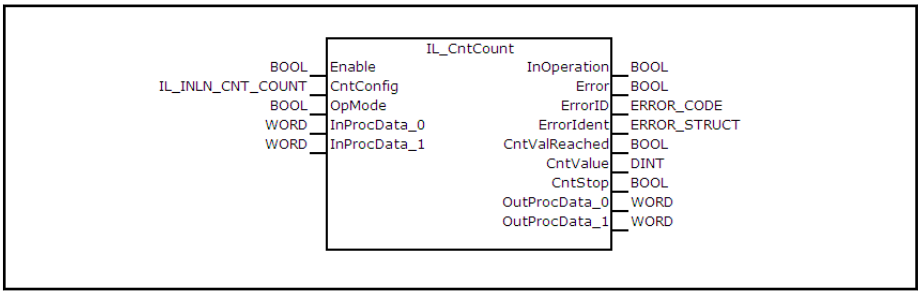


Fig. 14-26: IL_CntCount function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	CntConfig	IL_INLN_CNT_COUNT	Structure for counter configuration, see "IL_INLN_CNT_COUNT" on page 620
	Pause	BOOL	TRUE = Counting pause FALSE = Counting continued
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 624
	CntValReached	BOOL	TRUE - End value reached. The output behavior depends on the specifications made under "OutputBehavior"
	CntValue	DINT	Current counter value
	CntStop	BOOL	Counter is stopped/paused and "CntValue" contains the current counter value
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-79: IL_CntCount interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
Pause	BOOL	FALSE	TRUE	FALSE	Continuously with "Enable"

Tab. 14-80: Input behavior of the IL_CntCount

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

The parameterization of the **R-IB IL CNT** terminal as counter requires an assignment of the parameter structure **IL_INLN_CNT_COUNT** with suitable values.

The terminal is configured with a positive edge at the **Enable** input. The successful configuration is indicated by **InOperation** = TRUE and the valid counting values are available at the **CntValue** output.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Enable** input, errors are deleted and the terminal is reconfigured.

A positive edge at the **Pause** input can interrupt the counting at any point in time

A transition from 0 to 1 at the **CntStop** output indicates that the counter is in the "Stop" state. The current counter value can be read at the **CntValue** output.

The interrupted counting can be restarted with a positive edge at the **Pause** input.

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:

   Byte 0/1/2/3 = Plug 4/3/2/1.

2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

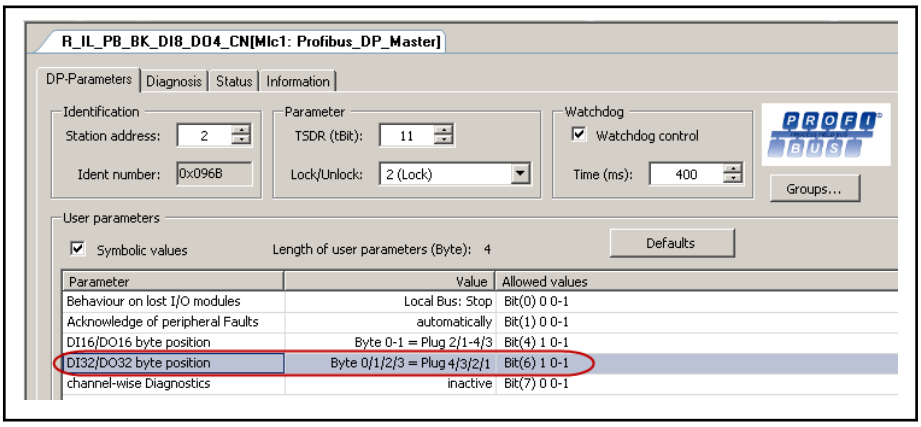


Fig. 14-27: Dialog: Bus terminal with changed byte sequence

IL_INLN_CNT_COUNT

The following structure allows the configuration of the counter terminal in the "event counting" mode.

Name	Type	Description	Min. value	Max. value	Default value	Effective
BusReset	BOOL	FALSE - Bus reset resets the output, stops all counting options and deletes the operation mode settings TRUE – No response on bus reset	FALSE	TRUE	FALSE	Positive edge at "Enable"
InConfig	IL_INLN_CNT_INCONFIG	Refer to "IL_INLN_CNT_INCONFIG" on page 621	NO_INPUT_CONFIG	SOURCE_GATE_OPERATION	NO_INPUT_CONFIG	Positive edge at "Enable"
SourceGateOperation	IL_INLN_CNT_SG_OPERATION	Refer to "IL_INLN_CNT_SG_OPERATION" on page 621	NONE	SOURCE_OR_GATE	NONE	Positive edge at "Enable"
PulseWidth	IL_INLN_CNT_PULSE_WIDTH	Pulse length with digital output, see "IL_INLN_CNT_PULSE_WIDTH" on page 622	PULSE_10MS	PULSE_1000MS	PULSE_10MS	Positive edge at "Enable"
CntRepeat	BOOL	FALSE - Unique counting TRUE - Permanent counting	FALSE	TRUE	FALSE	Positive edge at "Enable"

Name	Type	Description	Min. value	Max. value	Default value	Effective
Gate	IL_INLN_CNT_GATE	Refer to "IL_INLN_CNT_GATE" on page 622	NO_FUNCTION	CNT_AT_HIGH_RESET_ON_RISE	NO_FUNCTION	Positive edge at "Enable"
CntDir	BOOL	TRUE – Counting up FALSE – Counting down	FALSE	TRUE	FALSE	Positive edge at "Enable"
OutputBehavior	IL_INLN_CNT_OUTBEHAVE	Refer to "IL_INLN_CNT_OUTBEHAVE" on page 622	HIGH_PULSE	LOW_PULSE	HIGH_PULSE	Positive edge at "Enable"
InitialValue	DINT	Initial value, start counting (0-16777215)	0	16777215	0	Positive edge at "Enable"
FinalValue	DINT	End value, counting (0-16777215)	0	16777215	0	Positive edge at "Enable"

Tab. 14-81: Permitted values of IL_INLN_CNT_COUNT for IL_CntCount

IL_INLN_CNT_INCONFIG

An input filter or input logic to link two input signals can be configured using the enumeration.

ENUM	Description
NO_INPUT_CONFIG	No input configuration
FILTER_FOR_MECH_SWITCH	RC filter can be connected to source and gate. This allows the use of mechanical switches (e.g. relays)
SOURCE_GATE_OPERATION	Source Gate Logic Operation, a combination of source and gate signal can be selected as counter signal

IL_INLN_CNT_SG_OPERATION

Source-gate logic operation, see "IL_INLN_CNT_SG_OPERATION" on page 621

In case of the source-gate operation, one signal is generated by the two input signals Source (S) and Gate (G). After processing according to the logic function, this signal is available as new "Source" signal. It is called Source' (S') in the following. The original source signal is not available anymore.

The original gate signal can still be used.

The most common logic functions can be realized via this source-gate operation.

ENUM	Description
NONE	No input configuration
SOURCE_NOR_GATE	**"S' = S NOR G"**. The S' is formed from NOR gate between Source and Gate signals
SOURCE_AND_NOT_GATE	**"S' = S AND /G"**. S' is formed from AND gate between Source and inverter on the Gate signals
NOT_GATE	**"S' = /G"**. The S' is formed from inverter on the Gate signals
NOT_SOURCE_AND_GATE	**"S' = /S AND G"**. The S' is formed from AND gate between inverter on Source and Gate signals
NOT_SOURCE	**"S' = /S"**. The S' is formed from inverter on Source.
SOURCE_EXOR_GATE	**"S' = S EXOR G"**. The S' is formed from EXOR gate between Source and Gate signals

ENUM	Description
SOURCE_NAND_GATE	**“S’ = S NAND G”**. The S’ is formed from NAND gate between Source and Gate signals
SOURCE_AND_GATE	**“S’ = S AND G”**. The S’ is formed from AND gate between Source and Gate signals
SOURCE_EXNOR_GATE	**“S’ = S EXNOR G”**. The S’ is formed from EXNOR gate between Source and Gate signals
SOURCE	**“S’ = S”**. The S’ is formed from Source
SOURCE_OR_NOT_GATE	**“S’ = S OR /G”**. The S’ is formed from OR gate between Source and inverter on the Gate signals
GATE	**“S’ = G”**. The S’ is formed from Gate
NOT_SOURCE_OR_GATE	**“S’ = /S OR G”**. The S’ is formed from OR gate between inverter on Source and Gate signals
SOURCE_OR_GATE	**“S’ = S OR G”**. The S’ is formed from OR gate between Source and Gate signals

IL_INLN_CNT_GATE

The enumeration describes the condition at the "Gate" input that has to be met for counting.

ENUM	Description
NO_FUNCTION	No function
CNT_AT_HIGH	Counting at Gate if high level
CNT_AT_LOW	Counting at Gate if low level
CNT_ON_RISE	Start counting at a rising edge (0/1) at Gate
CNT_ON_FALL	Start counting at a falling edge (1/0) at Gate
CNT_AT_HIGH_RESET_ON_RISE	Count at Gate is high; The count value is reset by rising edge

IL_INLN_CNT_OUTBEHAVE

The enumeration defines the switching behavior of the digital output at "TerminalCount" (reaching the end value).

ENUM	Description
INACTIVE	Output not active
HIGH_PULSE	A positive pulse is generated
LOW_PULSE	A negative pulse is generated
TOGGLE_LOW	Previous state is inverted, initial LOW
TOGGLE_HIGH	Previous state is inverted, initial HIGH
HIGH	Output to HIGH
LOW	Output to LOW

IL_INLN_CNT_PULSE_WIDTH

The enumeration type specifies the pulse length of the digital output.

ENUM	Description
PULSEWIDTH_10MS	10 milliseconds
PULSEWIDTH_50MS	50 milliseconds
PULSEWIDTH_100MS	100 milliseconds
PULSEWIDTH_200MS	200 milliseconds

ENUM	Description
PULSEWIDTH_300MS	300 milliseconds
PULSEWIDTH_400MS	400 milliseconds
PULSEWIDTH_500MS	500 milliseconds
PULSEWIDTH_1000MS	1000 milliseconds

Example

Use this function block to make the system settings for the counter terminal in the "frequency measurement" mode.

System settings are basic settings for all terminal modes.

The system settings includes: Input configuration, source-gate operations, but reset behavior and pulse length parameters.

If the input configuration is selected with the logic source-gate operation, a logic operation to generate the source signal **S'** from the input source signal **S** and the gate signal **G** can be performed.

Due to the configuration of the counter terminal in this example, it repetitively counts from a starting value of 100 to an end value of 500.

The output behavior is specified with **HIGH_PULSE**, a 0/1 transition is generated when reaching the counter end value.

Enabling **Pause** stops the counting. Counting starts again upon a 1/0 transition at **Pause**.

Declaration of the example

```
PROGRAM PrgIL_CntCount
VAR
    fbIL_CntCount    : IL_CntCount ;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_1: WORD;
    wOutProcessData_0: WORD;
    diCntValue: DINT;
    bCntValReached: BOOL;
END_VAR
```

Implementation of the example

```
// System settings
 fbIL_CntCount.CntConfig.BusRese    := TRUE;
 fbIL_CntCount.CntConfig.Gate        := IL_INLN_CNT_GATE.CNT_AT_HIGH    ;
 fbIL_CntCount.CntConfig.InConfig     := IL_INLN_CNT_INCONFIG.SOURCE_GATE_OPERATION;
 fbIL_CntCount.CntConfig.SourceGateOperation := IL_INLN_CNT_SG_OPERATION.SOURCE;
 fbIL_CntCount.CntConfig.PulseWidth :=IL_INLN_CNT_PULSE_WIDTH.PULSEWIDTH_1000MS;

// Parameter settings for event count mode
 fbIL_CntCount.CntConfig.CntDir        := TRUE;
 fbIL_CntCount.CntConfig.CntRepeat    := TRUE;
 fbIL_CntCount.CntConfig.OutputBehavior    := IL_INLN_CNT_OUTBEHAVE.HIGH_PULSE;
 fbIL_CntCount.CntConfig.InitialValue    := DINT#100    ;
 fbIL_CntCount.CntConfig.FinalValue    := DINT#500;
 fbIL_CntCount.InProcData_0        := wInProcessData_0;
 fbIL_CntCount.InProcData_1        := wInProcessData_1;
 wOutProcessData_0            := fbIL_CntCount.OutProcData_0;
 wOutProcessData_1            := fbIL_CntCount.OutProcData_1;

 // FB call
   fbIL_CntCount(Enable:=TRUE);

 // Data read
 IF    fbIL_CntCount.InOperation = TRUE THEN
     diCntValue    := fbIL_CntCount.CntValue;
     bCntValReached := fbIL_CntCount.CntValReached;
 END_IF

 // Stopping the counter operation
 IF    fbIL_CntCount.InOperation = TRUE THEN
```

```
    fbIL_CntCount.Pause := TRUE ;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_CntCount".

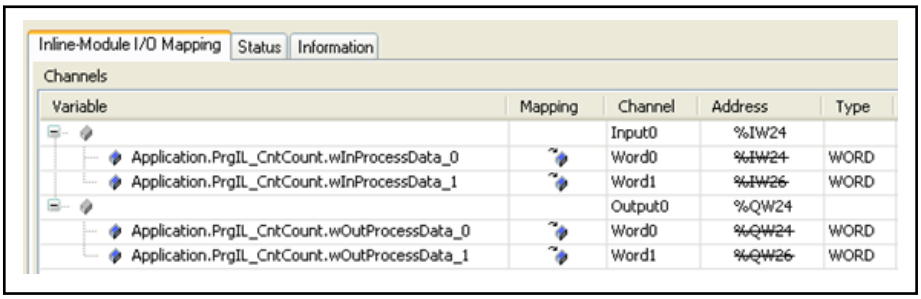


Fig. 14-28: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“InConfig” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“SourceGateOperation” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000003	“PulseWidth” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000004	“Gate” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000005	“OutputBehavior” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000006	“InitialValue” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000007	“FinalValue” is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Timeout, device does not response

Tab. 14-82: Error codes of the IL_CntCount

14.4.2 IL_CntFrequency

Brief description

The **IL_CntFrequency** function block parameterizes the **R-IB IL CNT** terminal (Inline counter terminal) in the "Frequency measurement" mode and reads the frequency (time- or state-controlled).

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

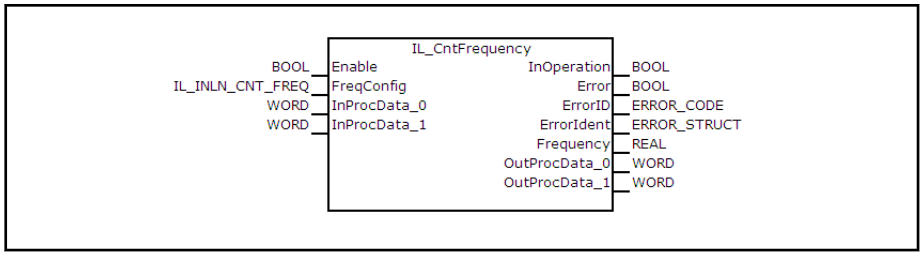


Fig. 14-29: IL_CntFrequency function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	FreqConfig	IL_INLN_CNT_FREQ	Structure for counter configuration, see "IL_INLN_CNT_FREQ" on page 626
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 630
	Frequency	REAL	In the time-controlled mode, the standardized frequency is output in [Hz]. Note: The values are only valid if the gate time (period during the measurement) elapsed. For the state-controlled mode, the counted pulses of the relevant measurement are output.
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-83: IL_CntFrequency interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous

Tab. 14-84: Input behavior of the IL_CntFrequency

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

The counter terminal is configured in the "Frequency" mode with a positive edge at "Enable".

The frequency can be either measured time- or state-controlled.

To parameterize the counter, all parameters have to be provided with corresponding values.

The terminal is configured with a positive edge at the **Enable** input. The successful configuration is indicated by **InOperation** = TRUE and the valid counting values are available at the **Frequency** output.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Enable** input, errors are deleted and the terminal is reconfigured.

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:

   Byte 0/1/2/3 = Plug 4/3/2/1.

2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

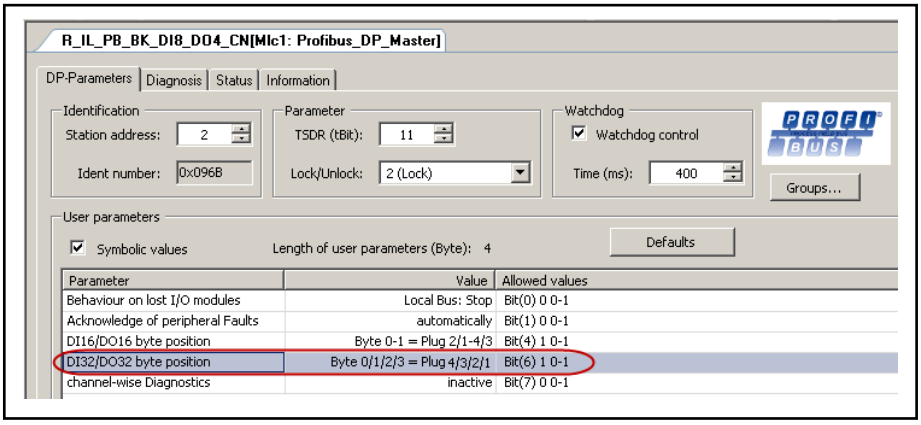


Fig. 14-30: Dialog: Bus terminal with changed byte sequence

IL_INLN_CNT_FREQ The structure supports the configuration of the counter terminal in the "Frequency measurement" mode.

Name	Type	Description	Min. value	Max. value	Default value	Effective
BusReset	BOOL	FALSE - Bus reset resets the output, stops all counting options and deletes the operation mode settings. TRUE – No response on bus reset.	FALSE	TRUE	FALSE	Continuously with "Enable"
InConfig	IL_INLN_CNT_INCONFIG	Refer to "IL_INLN_CNT_INCONFIG" on page 627	NO_INPUT_CONFIG	SOURCE_GATE_OPERATION	NO_INPUT_CONFIG	Positive edge at "Enable"
SourceGateOperation	IL_INLN_CNT_SG_OPERATION	Refer to "IL_INLN_CNT_SG_OPERATION" on page 627	NONE	SOURCE_OR_GATE	NONE	Positive edge at "Enable"
FreqControl	IL_INLN_CNT_FREQ_CONTROL	Refer to "IL_INLN_CNT_FREQ_CONTROL" on page 626	TIME_MEASURE	STATE_FALL_EDGE	TIME_MEASURE	Positive edge at "Enable"
Factor	INT	Gate time = Factor x 10 ms, Possible factors are 1, 2, 10, 50, 100, 500, 1000	0	1000	0	Positive edge at "Enable"

Tab. 14-85: Permitted values of IL_INLN_CNT_FREQ for IL_CntFrequency

IL_INLN_CNT_FREQ_CONTROL This enumeration contains options to measure the frequency.

ENUM	Description
TIME_MEASURE	Time-controlled frequency measurement, after the timeout of the gate time, the counter is reset to the initial value.
STATE_HIGH	With this measurement, counting takes place while the gate is HIGH. Counting stops when it changes to LOW. The last count value is applied to the input data. The next HIGH causes counting to restart from 0
STATE_LOW	With this measurement counting takes place while the gate is LOW. Counting stops when it changes to HIGH. The last count value is applied to the input data. The next LOW causes the count to restart from 0.
STATE_RISING_EDGE	With this measurement, counting begins immediately after the frequency measurement command was sent. The current count value is applied to the input data at each rising edge of the gate signal. The counter is reset to 0 and counting continues.
STATE_FALL_EDGE	With this measurement, counting begins immediately after the frequency measurement command was sent. The current count value is applied to the input data at each falling edge of the gate signal. The counter is reset to 0 and counting continues.

IL_INLN_CNT_INCONFIG An input filter or input logic to link two input signals can be configured using the enumeration.

ENUM	Description
NO_INPUT_CONFIG	No input configuration
FIL-TER_FOR_MECH_SWITCH	RC filter can be connected to source and gate. This allows the use of mechanical switches (e.g. relays)
SOURCE_GATE_OPERA-TION	Source Gate Logic Operation, a combination of source and gate signal can be selected as counter signal

Source-gate logic operation, see "IL_INLN_CNT_SG_OPERATION" on page 627

IL_INLN_CNT_SG_OPERATION In case of the source-gate operation, one signal is generated by the two input signals Source (S) and Gate (G). After processing according to the logic function, this signal is available as new "Source" signal. It is called Source' (S') in the following. The original source signal is not available anymore.

The original gate signal can still be used.

The most common logic functions can be realized via this source-gate operation.

ENUM	Description
NONE	No input configuration
SOURCE_NOR_GATE	**"S' = S NOR G"**. The S' is formed from NOR gate between Source and Gate signals
SOURCE_AND_NOT_GATE	**"S' = S AND /G"**. S' is formed from AND gate between Source and inverter on the Gate signals
NOT_GATE	**"S' = /G"**. The S' is formed from inverter on the Gate signals
NOT_SOURCE_AND_GATE	**"S' = /S AND G"**. The S' is formed from AND gate between inverter on Source and Gate signals

ENUM	Description
NOT_SOURCE	**“S’ = /S”**. The S’ is formed from inverter on Source.
SOURCE_EXOR_GATE	**“S’ = S EXOR G”**. The S’ is formed from EXOR gate between Source and Gate signals
SOURCE_NAND_GATE	**“S’ = S NAND G”**. The S’ is formed from NAND gate between Source and Gate signals
SOURCE_AND_GATE	**“S’ = S AND G”**. The S’ is formed from AND gate between Source and Gate signals
SOURCE_EXNOR_GATE	**“S’ = S EXNOR G”**. The S’ is formed from EXNOR gate between Source and Gate signals
SOURCE	**“S’ = S”**. The S’ is formed from Source
SOURCE_OR_NOT_GATE	**“S’ = S OR /G”**. The S’ is formed from OR gate between Source and inverter on the Gate signals
GATE	**“S’ = G”**. The S’ is formed from Gate
NOT_SOURCE_OR_GATE	**“S’ = /S OR G”**. The S’ is formed from OR gate between inverter on Source and Gate signals
SOURCE_OR_GATE	**“S’ = S OR G”**. The S’ is formed from OR gate between Source and Gate signals

Example Use this function block to make the system settings for the counter terminal in the "frequency measurement" mode.

System settings are basic settings for all terminal modes.

The system settings includes: Input configuration, source-gate operations, but reset behavior and pulse length parameters.

If the input configuration is selected with the logic source-gate operation, a logic operation to generate the source signal **S'** from the input source signal **S** and the gate signal **G** can be performed.

Time-controlled frequency measurement:

In case of time-controlled frequency measurement, the parameter "FreqControl" is used as factor to determine the gate time (period during the measurement) as a multiple of 10 ms. After the timeout of the gate time, the counter is reset to the initial value.

The project members have to select the optimum factor, as resolution and time are reciprocally proportional.

There are the following relations between the individual variables:

- Gate time = Factor x 10 ms
- Resolution = 1 / gate time
- Resolution = 1 / (factor x 10 ms)
- Frequency = Counting value x 100 / factor

Declaration of the example

```
PROGRAM prgIL_CntFrequency
VAR
    fbIL_CntFrequency: IL_CntFrequency;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    rFrequency: REAL;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_CntFrequency.FreqConfig.BusReset := TRUE ;
 fbIL_CntFrequency.FreqConfig.InConfig := IL_INLN_CNT_INCONFIG.SOURCE_GATE_OPERATION ;
```

```
fbIL_CntFrequency.FreqConfig.SourceGateOperation := IL_INLN_CNT_SG_OPERATION.SOURCE_AND_GATE ;
fbIL_CntFrequency.FreqConfig.FreqControl := IL_INLN_CNT_FREQ_CONTROL.TIME_MEASURE ;
fbIL_CntFrequency.FreqConfig.Factor := INT#50 ;
fbIL_CntFrequency.InProcData_0 := wInProcessData_0 ;
fbIL_CntFrequency.InProcData_1 := wInProcessData_1 ;
wOutProcessData_0 := fbIL_CntFrequency.OutProcData_0 ;
wOutProcessData_1 := fbIL_CntFrequency.OutProcData_1 ;

// FB call
 fbIL_CntFrequency(Enable:= TRUE);

// Data read
IF    fbIL_CntFrequency.InOperation = TRUE THEN
    rFrequency := fbIL_CntFrequency.Frequency ;
END_IF
```

State-controlled frequency measurement:

In case of the state-controlled frequency measurement, the parameter "FreqControl" indicates the state of the "Gate" input at which it is counted or the counting value is applied.

In this example, "FreqControl" is selected for **STATE_HIGH** (measurement runs until the gate is in the state **HIGH**.

Counting stops when the gate changes to **LOW**. The last counter value is applied as input value.

The subsequent **HIGH** starts the counting again from 0.

Declaration of the example

```
PROGRAM prgIL_CntFrequency
VAR
    fbIL_CntFrequency: IL_CntFrequency;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    rFrequency: REAL;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_CntFrequency.FreqConfig.BusReset := TRUE ;
 fbIL_CntFrequency.FreqConfig.InConfig := IL_INLN_CNT_INCONFIG.SOURCE_GATE_OPERATION ;
 fbIL_CntFrequency.FreqConfig.SourceGateOperation := IL_INLN_CNT_SG_OPERATION.SOURCE ;
 fbIL_CntFrequency.FreqConfig.FreqControl := IL_INLN_CNT_FREQ_CONTROL.STATE_HIGH ;
 fbIL_CntFrequency.InProcData_0 := wInProcessData_0 ;
 fbIL_CntFrequency.InProcData_1 := wInProcessData_1 ;
 wOutProcessData_0 := fbIL_CntFrequency.OutProcData_0 ;
 wOutProcessData_1 := fbIL_CntFrequency.OutProcData_1 ;

 // FB call
 fbIL_CntFrequency(Enable:= TRUE);

 // Data read
 IF    fbIL_CntFrequency.InOperation = TRUE THEN
     rFrequency := fbIL_CntFrequency.Frequency ;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_CntFrequency".

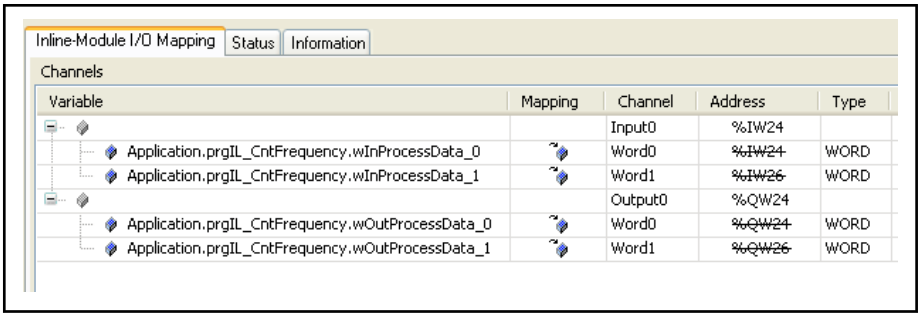


Fig. 14-31: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“InConfig” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“SourceGateOperation” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000003	“FreqControl” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000004	“Factor” is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response

Tab. 14-86: Error codes of the IL_CntFrequency

14.4.3 IL_CntPulse

Brief description

The **IL_CntPulse** function block parameterizes the **R-IB IL CNT** terminal (Inline counter terminal) in the "Pulse generator" mode and generates pulses.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

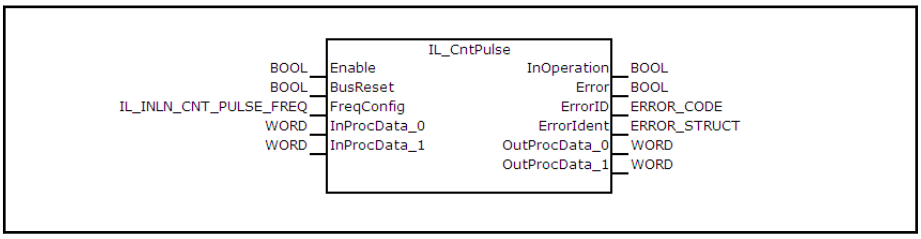


Fig. 14-32: IL_CntPulse function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	BusReset	BOOL	FALSE - Bus reset resets the output, stops all counting options and deletes the operation mode settings. TRUE – No response on bus reset.
	Frequency	IL_INLN_CNT_PULSE_FREQ	Frequency factor in 500 Hz increments, see "IL_INLN_CNT_PULSE_FREQ" on page 632 `PulseFreq = 1000 Hz + (Factor × 500 Hz)`
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 633
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-87: IL_CntPulse interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
BusReset	BOOL	FALSE	TRUE	FALSE	Positive edge at "Enable"
Frequency	IL_INLN_CNT_PULSE_FREQ	FREQ_1000HZ	FREQ_10000HZ	FREQ_1000HZ	Positive edge at "Enable"

Tab. 14-88: Input behavior of the IL_CntPulse

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

The counter terminal is configured as pulse generator with a positive edge at **Enable**.

The pulse generator can generate frequencies between 1 kHz and 10 kHz in increments of 500 Hz.

The terminal is configured with a positive edge at the **Enable** input. The configuration is completed successfully if **InOperation** = TRUE. Pulse generation starts.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Enable** input, errors are deleted and the terminal is reconfigured.

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:
   Byte 0/1/2/3 = Plug 4/3/2/1.
2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

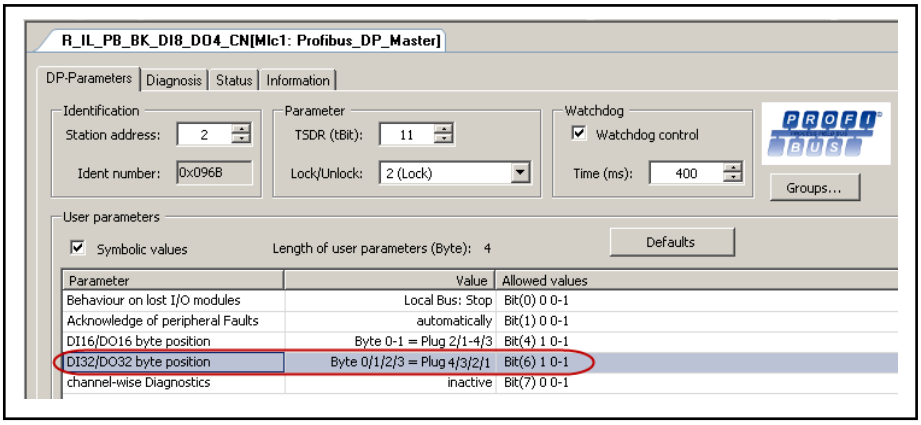


Fig. 14-33: Dialog: Bus terminal with changed byte sequence

IL_INLN_CNT_PULSE_FREQ

The enumeration describes the possible frequency parameters: factor, command frequency F_{com}, actual frequency F_{act} and errors during the pulse generation.

```
Pulse_Freq = 1000 Hz + (Factor × 500 Hz)
```

ENUM	Description
FREQ_1000HZ	Factor = 0, Fcomm = 1000 Hz, Fact = 1000 Hz, error = 0%
FREQ_1500HZ	Factor = 1, Fcomm = 1500 Hz, Fact = 1495 Hz, error = -0.2%
FREQ_2000HZ	Factor = 2, Fcomm = 2000 Hz, Fact = 2000 Hz, error = 0%
FREQ_2500HZ	Factor = 3, Fcomm = 2500 Hz, Fact = 2500 Hz, error = 0%
FREQ_3000HZ	Factor = 4, Fcomm = 3000 Hz, Fact = 3012 Hz, error = 0.4%
FREQ_3500HZ	Factor = 5, Fcomm = 3500 Hz, Fact = 3521 Hz, error = 0.6%
FREQ_4000HZ	Factor = 6, Fcomm = 4000 Hz, Fact = 4000 Hz, error = 0%
FREQ_4500HZ	Factor = 7, Fcomm = 4500 Hz, Fact = 4505 Hz, error = 0.11%
FREQ_5000HZ	Factor = 8, Fcomm = 5000 Hz, Fact = 5000 Hz, error = 0%
FREQ_5500HZ	Factor = 9, Fcomm = 5500 Hz, Fact = 5495 Hz, error = -0.09%
FREQ_6000HZ	Factor = 10, Fcomm = 6000 Hz, Fact = 5998 Hz, error = -0.2%
FREQ_6500HZ	Factor =11, Fcomm = 6500 Hz, Fact = 6494 Hz, error = -0.09%
FREQ_7000HZ	Factor = 12, Fcomm = 7000 Hz, Fact = 6993 Hz, error = -0.01%
FREQ_7500HZ	Factor = 13, Fcomm = 7500 Hz, Fact = 7519 Hz, error = 0.25%
FREQ_8000HZ	Factor = 14, Fcomm = 8000 Hz, Fact = 8000 Hz, error = 0%
FREQ_8500HZ	Factor = 15, Fcomm = 8500 Hz, Fact = 8475 Hz, error = -0.29%
FREQ_9000HZ	Factor = 16, Fcomm = 9000 Hz, Fact = 9009 Hz, error = 0.1%

ENUM	Description
FREQ_9500HZ	Factor = 17, Fcomm = 9500 Hz, Fact = 9525 Hz, error = 0.25%
FREQ_10000HZ	Factor = 18, Fcomm = 10000 Hz, Fact = 10000 Hz, error = 0%

Example

The function block configures the counter terminal as pulse generator.

The pulse generator can produce frequencies between 1 kHz and 10 kHz in increments of 500 Hz.

In this example, pulses of a frequency of 1kHz can be generated due to the function block configuration.

Declaration of the example

```
PROGRAM PrgIL_CntPulse
VAR
    fbIL_CntPulse: IL_CntPulse;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_CntPulse.BusReset := TRUE ;
 fbIL_CntPulse.Frequency := IL_INLN_CNT_PULSE_FREQ.FREQ_1000HZ;
 fbIL_CntPulse.InProcData_0 := wInProcessData_0 ;
 fbIL_CntPulse.InProcData_1 := wInProcessData_1 ;
 wOutProcessData_0 := fbIL_CntPulse.OutProcData_0 ;
 wOutProcessData_1 := fbIL_CntPulse.OutProcData_1 ;

 // FB call
   fbIL_CntPulse(Enable:=TRUE);
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_CntPulse".

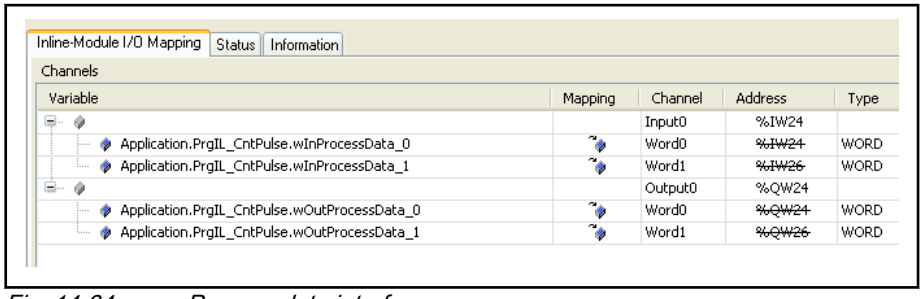


Fig. 14-34: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“Frequency” input is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response

Tab. 14-89: Error codes of the IL_CntPulse

14.4.4 IL_CntTime

Brief description

The **IL_CntTime** function block parameterizes the **R-IB IL CNT** terminal (Inline counter terminal) in the "Time measurement" mode and reads the counted values (period length and pulse duration).

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

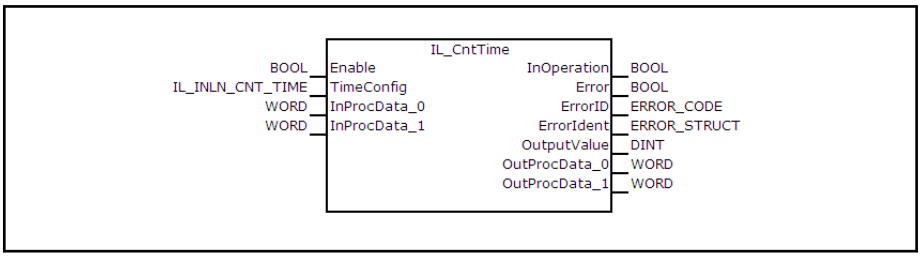


Fig. 14-35: IL_CntTime function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	TimeConfig	IL_INLN_CNT_TIME	Structure for counter configuration, see "IL_INLN_CNT_TIME" on page 635
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 639
	OutputValue	DINT	Measured value depends on the resolution: Value ranges: for 2 µs, 0 ... 131070 µs, for 2 ms, 0 ... 131070 ms and for 10 ms, 0 ... 655350 ms
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-90: IL_CntTime interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous

Tab. 14-91: Input behavior of the IL_CntTime

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

The parameterization of the **R-IB IL CNT** terminal in the "Time measurement" mode requires an assignment of the parameter structure **IL_INLN_CNT_TIME** with suitable values.

The terminal is configured with a positive edge at the **Enable** input. The successful configuration is indicated by **InOperation** = TRUE and the valid values are available at the **OutputValue** output.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Enable** input, errors are deleted and the terminal is reconfigured.

As specified in the **MeasureLength** input, the period length or pulse duration can be measured.

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:

   Byte 0/1/2/3 = Plug 4/3/2/1.
2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

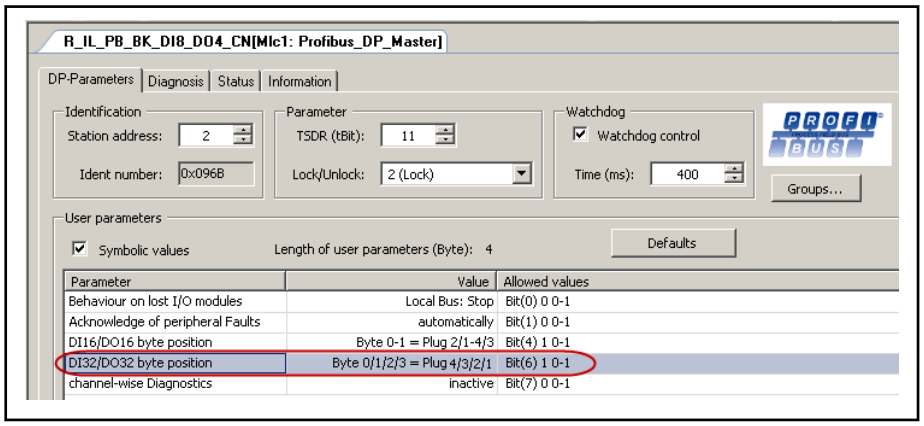


Fig. 14-36: Dialog: Bus terminal with changed byte sequence

IL_INLN_CNT_TIME

The structure supports the configuration of the counter terminal in the "Time measurement" mode.

Name	Type	Description	Min. value	Max. value	Default value	Effective
BusReset	BOOL	FALSE - Bus reset resets the output, stops all counting options and deletes the operation mode settings TRUE – No response on bus reset	FALSE	TRUE	FALSE	Positive edge at "Enable"
InConfig	IL_INLN_CNT_INCONFIG	Refer to "IL_INLN_CNT_INCONFIG" on page 636	NO_INPUT_CONFIG	SOURCE_GATE_OPERATION	NO_INPUT_CONFIG	Positive edge at "Enable"
SourceGateOperation	IL_INLN_CNT_SG_OPERATION	Refer to "IL_INLN_CNT_SG_OPERATION" on page 636	NONE	SOURCE_OR_GATE	NONE	Positive edge at "Enable"

Name	Type	Description	Min. value	Max. value	Default value	Effective
Resolution	IL_INLN_CNT_TIME_RES	Refer to "IL_INLN_CNT_TIME_RES" on page 637	RES_2US	RES_10MS	RES_2MS	Positive edge at "Enable"
Measure-Length	BOOL	FALSE – Measuring the period length TRUE – Measuring the pulse duration	FALSE	TRUE	FALSE	Positive edge at "Enable"
OutputRelation	IL_INLN_CNT_TIME_OUT_RELATION	Comparative conditions, see "IL_INLN_CNT_TIME_OUT_RELATION" on page 637	NO_RELATION	CV_LT_IV_HYSTERESIS	NO_RELATION	Positive edge at "Enable"
InitialValue	DINT	Initial value depends on the resolution: 2 µs- 0 ... 131000 2 ms- 0 ... 131000 10 ms- 0 ... 655350	0	Resolution 2 µs: 131000 2 ms: 131000 10 ms: 655000	0	Positive edge at "Enable"
FinalValue	DINT	End value depends on the resolution: 2 µs- 0 ... 131000 2 ms- 0 ... 131000 10 ms- 0 ... 655350	0	Resolution 2 µs: 131000 2 ms: 131000 10 ms: 655000	0	Positive edge at "Enable"

Tab. 14-92: Permitted values of IL_INLN_CNT_COUNT for IL_CntTime

IL_INLN_CNT_INCONFIG An input filter or input logic to link two input signals can be configured using the enumeration.

ENUM	Description
NO_INPUT_CONFIG	No input configuration
FILTER_FOR_MECH_SWITCH	RC filter can be connected to source and gate. This allows the use of mechanical switches (e.g. relays)
SOURCE_GATE_OPERATION	Source Gate Logic Operation, a combination of source and gate signal can be selected as counter signal

Source-gate logic operation, see "IL_INLN_CNT_SG_OPERATION" on page 636

IL_INLN_CNT_SG_OPERATION In case of the source-gate operation, one signal is generated by the two input signals Source (S) and Gate (G). After processing according to the logic function, this signal is available as new "Source" signal. It is called Source' (S') in the following. The original source signal is not available anymore.

The original gate signal can still be used.

The most common logic functions can be realized via this source-gate operation.

ENUM	Description
NONE	No input configuration
SOURCE_NOR_GATE	**"S' = S NOR G"**. The S' is formed from NOR gate between Source and Gate signals
SOURCE_AND_NOT_GATE	**"S' = S AND /G"**. S' is formed from AND gate between Source and inverter on the Gate signals
NOT_GATE	**"S' = /G"**. The S' is formed from inverter on the Gate signals
NOT_SOURCE_AND_GATE	**"S' = /S AND G"**. The S' is formed from AND gate between inverter on Source and Gate signals
NOT_SOURCE	**"S' = /S"**. The S' is formed from inverter on Source.
SOURCE_EXOR_GATE	**"S' = S EXOR G"**. The S' is formed from EXOR gate between Source and Gate signals
SOURCE_NAND_GATE	**"S' = S NAND G"**. The S' is formed from NAND gate between Source and Gate signals
SOURCE_AND_GATE	**"S' = S AND G"**. The S' is formed from AND gate between Source and Gate signals
SOURCE_EXNOR_GATE	**"S' = S EXNOR G"**. The S' is formed from EXNOR gate between Source and Gate signals
SOURCE	**"S' = S"**. The S' is formed from Source
SOURCE_OR_NOT_GATE	**"S' = S OR /G"**. The S' is formed from OR gate between Source and inverter on the Gate signals
GATE	**"S' = G"**. The S' is formed from Gate
NOT_SOURCE_OR_GATE	**"S' = /S OR G"**. The S' is formed from OR gate between inverter on Source and Gate signals
SOURCE_OR_GATE	**"S' = S OR G"**. The S' is formed from OR gate between Source and Gate signals

IL_INLN_CNT_TIME_RES

The enumeration type describes the resolution of **all** values per LSB (low-order bit).

ENUM	Description
RES_2US	2 microseconds
RES_2MS	2 milliseconds
RES_10MS	10 milliseconds

IL_INLN_CNT_TIME_OUT_RELATION

The enumeration type describes the output comparative conditions.

☞ It depends on the condition whether a limit value is included in the condition. Internally, each condition is based on a comparative counting value lower than the starting value or/an higher than the end value.

ENUM	Description
NO_RELATION	No relation condition
CV_GE_IV	Count value greater than or equal to initial value
CV_LT_IV	Count value less than initial value
CV_WITHIN_IV_AND_FV	Count value within initial and final value
CV_OUTSITE_IV_AND_FV	Count value outside initial and final value

ENUM	Description
CV_GT_FV	Count value greater than final value
CV_LT_IV_HYSTERESIS	Count value less than initial value with hysteresis

Example The function block is required to make the system settings for the counter terminal in the "Time measurement" mode.

System settings are basic settings for all terminal modes.

The system settings includes: Input configuration, source-gate operations, but reset behavior and pulse length parameters.

If the input configuration is selected with the logic source-gate operation, a logic operation to generate the source signal **S'** from the input source signal **S** and the gate signal **G** can be performed.

Measurement starts at a positive edge (0/1 transition).

Measuring the pulse duration ends with the 1/0 transition, measuring the period length ends with the next 0/1 transition.

Only if the measurement is complete, the counter value is accepted for the process data.

If there is no countable transition, the measurement ends in a "TimeOut" and the "OutputValue" output is then deleted.

In this example, the function block is configured to measure the period length.

RES_2MS is selected as resolution. A period length between 2 ms and 131 s can be measured with this resolution.

InitialValue is 10 and **FinalValue** 100.

CV_GT_FV is selected as **OutputRelation**. Thus, if the counting value is higher than the end value with hysteresis, the output **OUT** is TRUE.

Declaration of the example

```
PROGRAM PrgIL_CntTime
VAR
    fbIL_CntTime: IL_CntTime;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    diPulsePeriod: DINT;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_CntTime.TimeConfig.BusReset := TRUE;
 fbIL_CntTime.TimeConfig.InConfig := IL_INLN_CNT_INCONFIG.SOURCE_GATE_OPERATION;
 fbIL_CntTime.TimeConfig.SourceGateOperation := IL_INLN_CNT_SG_OPERATION.SOURCE_OR_GATE ;
 fbIL_CntTime.TimeConfig.Resolution := IL_INLN_CNT_TIME_RES.RES_2MS ;
 fbIL_CntTime.TimeConfig.InitialValue := DINT#10 ;
 fbIL_CntTime.TimeConfig.FinalValue := DINT#100 ;
 fbIL_CntTime.TimeConfig.MeasureLength := FALSE ;
 fbIL_CntTime.TimeConfig.OutputRelation := IL_INLN_CNT_TIME_OUT_RELATION.CV_GT_FV ;
 fbIL_CntTime.InProcData_0 := wInProcessData_0 ;
 fbIL_CntTime.InProcData_1 := wInProcessData_1 ;
 wOutProcessData_0 := fbIL_CntTime.OutProcData_0 ;
 wOutProcessData_1 := fbIL_CntTime.OutProcData_1 ;

 // FB call
   fbIL_CntTime(Enable:=TRUE);

 // Data read
 IF    fbIL_CntTime.InOperation = TRUE THEN
     diPulsePeriod := fbIL_CntTime.OutputValue ;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_CntTime".

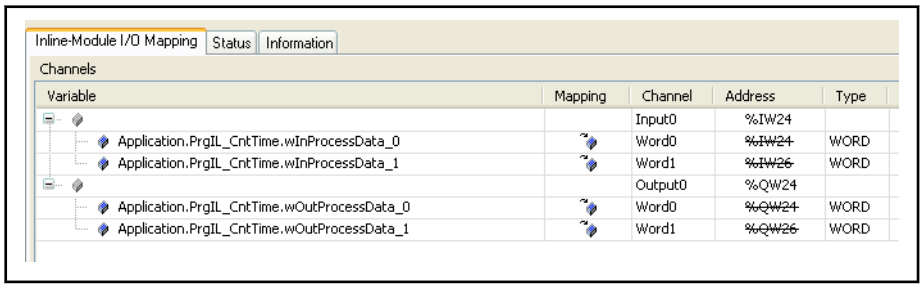


Fig. 14-37: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“InConfig” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“SourceGateOperation” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000003	“Resolution” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000004	“OutputRelation” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000005	“InitialValue” input is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000006	“FinalValue” input is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response

Tab. 14-93: Error codes of the IL_CntTime

14.4.5 IL_Pwm2Frequency

Brief description

The function block **IL_Pwm2Frequency** parameterizes the **R-IB IL PWM/2-PAC** terminal (Inline terminal for pulse width and frequency modulation) in the operation mode **Frequency generator with constant duty cycle** to generate output pulse sequences at the terminal.

One function block instance is required for each channel of the Inline I/O terminal. Both channels operate individually.



Except for the operation mode "Pulse direction signal", an own operation mode can be selected for each channel. If the terminal operates in the "Pulse direction signal" mode, both outputs are required.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

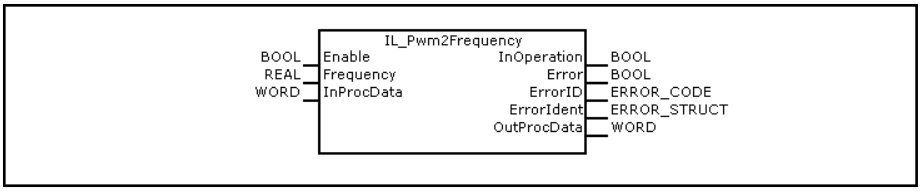


Fig. 14-38: IL_Pwm2Frequency function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	Frequency	REAL	Frequency, 12.21Hz ... 50,000 Hz, step width 12.21 Hz.
	InProcData	WORD	Process input word
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 641
	OutProcData	WORD	Process output word

Tab. 14-94: IL_Pwm2Frequency interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
Frequency	REAL	12.21 Hz	50,000 HZ	12.21 Hz	Continuously with "Enable"

Tab. 14-95: Input behavior of the IL_Pwm2Frequency

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

The function block configures the channel in the "Frequency generator" mode with a specified frequency and a constant 50% pulse-pause ratio.

The operation mode supports frequencies between 12.21 Hz and 50 kHz, with a step width of 12.21 Hz.

The configuration is completed successfully if **InOperation** = TRUE. Pulse generation starts. Continuous pulses are generated.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Enable** input, errors are deleted and the terminal is reconfigured.

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:

   Byte 0/1/2/3 = Plug 4/3/2/1.
2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

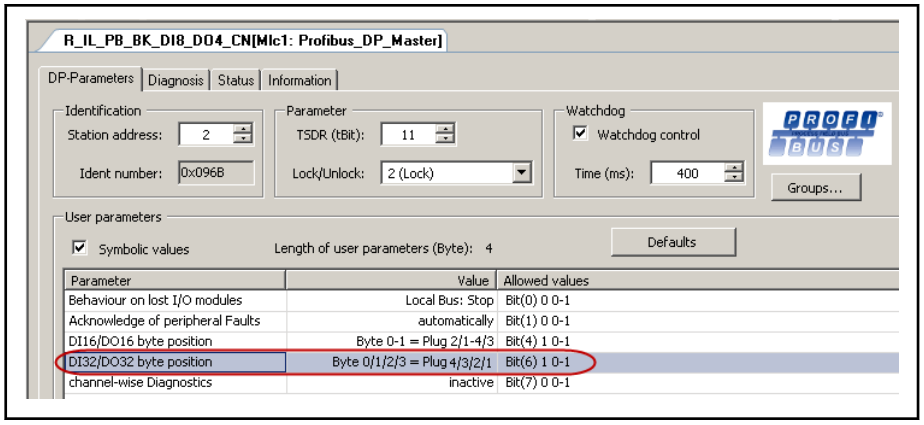


Fig. 14-39: Dialog: Bus terminal with changed byte sequence

Example

The function block is used to configure a PWM terminal in the **Frequency generator with constant duty cycle** mode. The frequency has to be specified and process data has to be mapped.

In this example, the function block **Pwm2Frequency** is configured as single channel for the user-defined frequency (and a pulse-pause ratio of 50%).

Declaration of the example

```
PROGRAM PrgIL_Pwm2Frequency
VAR
    fbIL_Pwm2Frequency: IL_Pwm2Frequency;
    wInProcessData: WORD;
    wOutProcessData: WORD;
END_VAR
```

Implementation of the example

```
// Configuration
  fbIL_Pwm2Frequency.Frequency   := REAL#50;
  fbIL_Pwm2Frequency.InProcData  := wInProcessData;
  wOutProcessData                := fbIL_Pwm2Frequency.OutProcData;

 // FB call
  fbIL_Pwm2Frequency(Enable := TRUE);

 // Frequency change during InOperation
  IF fbIL_Pwm2Frequency.InOperation = TRUE THEN
      fbIL_Pwm2Frequency.Frequency  := REAL#100;
  END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData" and "wOutProcessData" are declared in the program "prgIL_Pwm2Frequency".

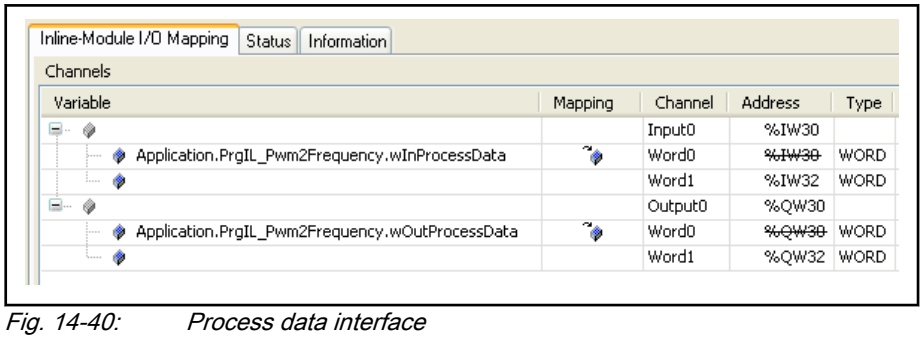


Fig. 14-40: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“Frequency” input is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response

Tab. 14-96: Error codes of the IL_Pwm2Frequency

14.4.6 IL_Pwm2Pwm

Brief description

The function block **IL_Pwm2Pwm** parameterizes the **R-IB IL PWM/2-PAC** terminal (Inline terminal for pulse width and frequency modulation) in the operation mode **Pulse width modulation (PWM) with variable duty cycle** to generate output pulse sequences at the terminal

One function block instance is required for each channel of the Inline I/O terminal. Both channels operate individually.

Except for the operation mode "Pulse direction signal", an own operation mode can be selected for each channel. If the terminal operates in the "Pulse direction signal" mode, both outputs are required.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

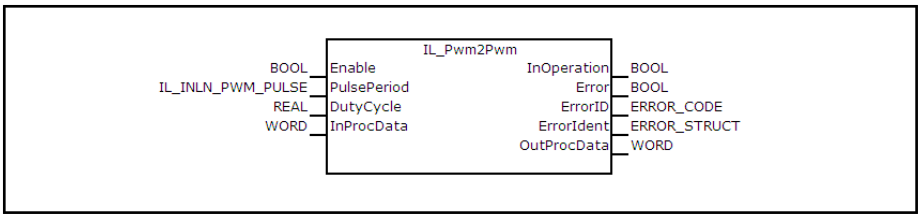


Fig. 14-41: IL_Pwm2Pwm function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	PulsePeriod	IL_INLN_PWM_PULSE	Pulse period, see "IL_INLN_PWM_PULSE" on page 644
	DutyCycle	REAL	Duty cycle of the PWM output in percent. The value can be specified between 0% and 99,45% with an increment of 0,39%. e.g. 50% ≈ 49,92%. 0% stops the PWM function.
	InProcData	WORD	Process input word

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 645
	OutProcData	WORD	Process output word

Tab. 14-97: IL_Pwm2Pwm interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
PulsePeriod	ENUM	PULSE_100US	PULSE_10S	PULSE_100US	Positive edge at "Enable"
DutyCycle	REAL	0	99.45	50	Continuously with "Enable"

Tab. 14-98: Input behavior of the IL_Pwm2Pwm

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Enable** input.

The function block configures the channel in pulse width modulation with the specified **PulsePeriod** and **DutyCycle** (pulse-pause ratio).

The period length can be specified between 100 µs and 10 s. This covers a frequency range between 10 kHz and 0,1 Hz.

A duty cycle between 0,39% and 99,45% can be selected.

The configuration is completed successfully if **InOperation** = TRUE. Pulse generation starts. Continuous pulses are generated.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Enable** input, errors are deleted and the terminal is reconfigured.



1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:

   Byte 0/1/2/3 = Plug 4/3/2/1.
2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

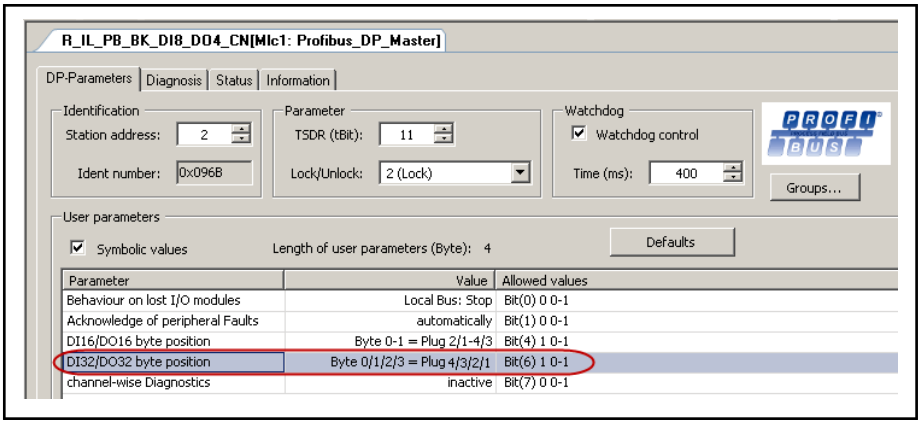


Fig. 14-42: Dialog: Bus terminal with changed byte sequence

IL_INLN_PWM_PULSE

The enumeration defines the pulse period of the PWM. Between 100 ms and 10 s can be specified. This results in a frequency range between 10 kHz and 0,1 Hz.

ENUM	Description
PULSE_100US	100 microseconds
PULSE_200US	200 microseconds
PULSE_400US	400 microseconds
PULSE_600US	600 microseconds
PULSE_800US	800 microseconds
PULSE_1MS	1 millisecond
PULSE_2MS	2 milliseconds
PULSE_4MS	4 milliseconds
PULSE_6MS	6 milliseconds
PULSE_8MS	8 milliseconds
PULSE_10MS	10 milliseconds
PULSE_20MS	20 milliseconds
PULSE_40MS	40 milliseconds
PULSE_60MS	60 milliseconds
PULSE_80MS	80 milliseconds
PULSE_100MS	100 milliseconds
PULSE_200MS	200 milliseconds
PULSE_400MS	400 milliseconds
PULSE_600MS	600 milliseconds
PULSE_800MS	800 milliseconds
PULSE_1S	1 second
PULSE_2S	2 seconds
PULSE_4S	4 seconds
PULSE_6S	6 seconds

ENUM	Description
PULSE_8S	8 seconds
PULSE_10S	10 seconds

Example

The function block is used to configure a PWM terminal in the "Pulse width modulation" mode. The settings contain **PulsePeriod** and **DutyCycle**. Process data has to be mapped.

In this example, the function block **Pwm2Pwm** is configured as single channel for a frequency of 10 ms and a pulse-pause ratio of 50%.

Declaration of the example

```
PROGRAM PrgIL_Pwm2Pwm
VAR
    fbIL_Pwm2Pwm: IL_Pwm2Pwm;
    wInProcessData: WORD;
    wOutProcessData: WORD;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_Pwm2Pwm.PulsePeriod := IL_INLN_PWM_PULSE.PULSE_10MS;
 fbIL_Pwm2Pwm.DutyCycle   := REAL#50.0;
 fbIL_Pwm2Pwm.InProcData  := wInProcessData;
 wOutProcessData          := fbIL_Pwm2Pwm.OutProcData;

 // FB call
  fbIL_Pwm2Pwm(Enable := TRUE);
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData" and "wOutProcessData" are declared in the program "prgIL_Pwm2Pwm".

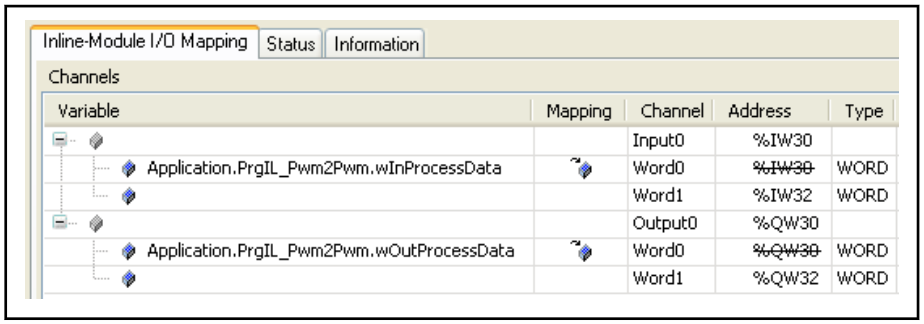


Fig. 14-43: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“PulsePeriod” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“DutyCycle” is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response

Tab. 14-99: Error codes of the IL_Pwm2Pwm

14.4.7 IL_Pwm2PulseDirection

Brief description

The function block **IL_Pwm2PulseDirection** parameterizes the **R-IB IL PWM/2-PAC** terminal (Inline terminal for pulse width and frequency modulation) in the operation mode **Pulse direction signal**.

Both outputs of the terminal are used together in this operation mode. Thus, only one channel is available.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

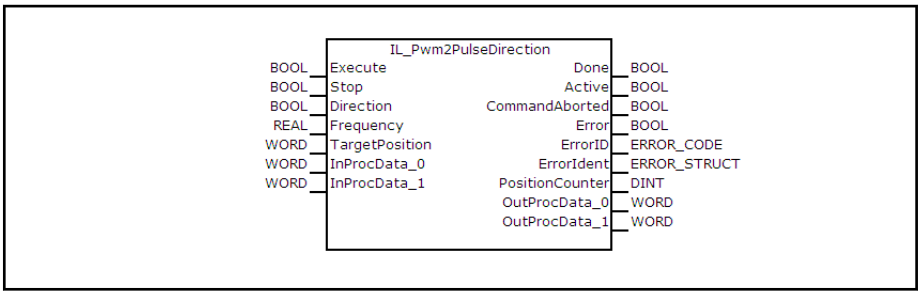


Fig. 14-44: IL_Pwm2PulseDirection function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	When the edge is positive, function block processing starts
	Stop	BOOL	Stops the pulse output and sets the position counter value to 0
	Direction	BOOL	Direction and output DO2 FALSE = Downwards or left TRUE = Upwards or right
	Frequency	REAL	Frequency, 12.21 Hz ... 25kHz, step width 12.21Hz.
	TargetPosition	WORD	Target position The target position has a value range between 16#0 an 16#FFFE (from 0 to 65534). The value 16#FFFF (65535dez) causes and endless pulse output. If the value is between 16#0 and 16#FFFE, the pulse output ends when the low-order 16 bits of the position counter are equal to the target position.
	InProcData0	WORD	Process input word 0
	InProcData1	WORD	Process input word 1

	Name	Type	Description
VAR OUTPUT	Done	BOOL	End of pulse sequence reached
	Active	BOOL	Function block operating, pulse sequence generated
	CommandAborted	BOOL	Function block processing aborted
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 649
	PositionCounter	DINT	Position counter The position counter counts the pulses output up to now depending on the signal "Direction" up or down.
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-100: IL_Pwm2PulseDirection interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Stop	BOOL	FALSE	TRUE	FALSE	Continuous
Direction	BOOL	FALSE	TRUE	FALSE	Positive edge at "Execute"
Frequency	REAL	12.21	25000	12.21	Continuous
TargetPosition	WORD	0	65535	0	Positive edge at "Execute"

Tab. 14-101: Input behavior of the IL_Pwm2PulseDirection

Functional description

The function block configures the terminal in the "Pulse direction signal" mode, specifying the frequency **Frequency** and the target position **TargetPosition**.

In this operation mode, the terminal outputs pulses with a frequency between 12,21 Hz ... 25 KHz, step width 12,21 Hz.

If **Frequency** = 0, the running pulse output is canceled.

The pulse-pause ratio is constantly at 50%.

The position counter counts the steps made according to **Direction** up or down to be able to read the drive position.

Done indicates whether the end of the pulse sequence is reached.

The target position **TargetPosition** is a value range between 16#0 and 16#FFFE (from 0 to 65534).

The value 16#FFFF (65535dez) causes and endless pulse output.

If the value is between 16#0 and 16#FFFE, the pulse output ends when the low-order 16 bits of the position counter are equal to the target position.

A positive edge at **Stop** stops the pulse output and sets the position counter value to 0.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Execute** input, errors are deleted and the terminal is reconfigured.

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:
   Byte 0/1/2/3 = Plug 4/3/2/1.
2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

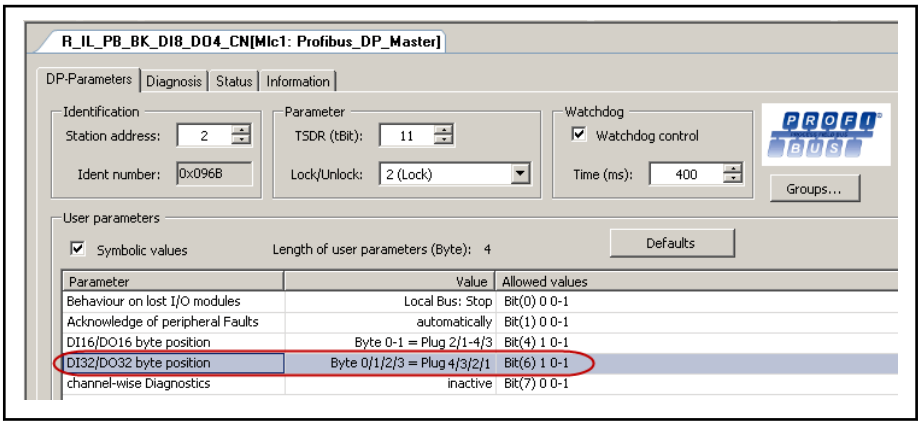


Fig. 14-45: Dialog: Bus terminal with changed byte sequence

Example The function block **IL_Pwm2PulseDirection** parameterizes the **R-IB IL PWM/2-PAC** terminal in the operation mode **Pulse direction signal**.

The settings contain **Frequency**, **TargetPosition** and **Direction**. Process data has to be mapped.

In this example, the function block **Pwm2PulseDirection** is configured for a drive with a frequency of 100 Hz and a target position of 2000 pulses.

Both outputs of the terminal are used together in this operation mode. Thus, only one channel is available.

The function block is enabled via a positive edge at **Execute**.

The function block configures the terminal and sets **Active** to TRUE.

The terminal starts with the pulse output. After completion, **Done** is set to TRUE and **Active** to FALSE.

Declaration of the example

```
PROGRAM PrgIL_Pwm2PulseDirection
VAR
    fbIL_Pwm2PulseDirection: IL_Pwm2PulseDirection;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcData_0: WORD;
    wOutProcData_1: WORD;
    diPosCount: DINT;
    bDone: BOOL;
END_VAR
```

Implementation of the example

```
// Configuration
   fbIL_Pwm2PulseDirection.Frequency      := REAL#100;
   fbIL_Pwm2PulseDirection.Direction      := TRUE;
   fbIL_Pwm2PulseDirection.TargetPosition := WORD#2000;
   fbIL_Pwm2PulseDirection.Stop           := FALSE;
   fbIL_Pwm2PulseDirection.InprocData_0   := wInProcessData_0;
   fbIL_Pwm2PulseDirection.InprocData_1   := wInProcessData_1;
   wOutProcData_0                         := fbIL_Pwm2PulseDirection.OutProcData_0;
   wOutProcData_1                         := fbIL_Pwm2PulseDirection.OutProcData_1;
```

```
// FB call
  fbIL_Pwm2PulseDirection(Execute := TRUE);

// To find the position value
  IF fbIL_Pwm2PulseDirection.Active = TRUE THEN
    diPosCount := fbIL_Pwm2PulseDirection.PositionCounter;
  END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_Pwm2PulseDirection".

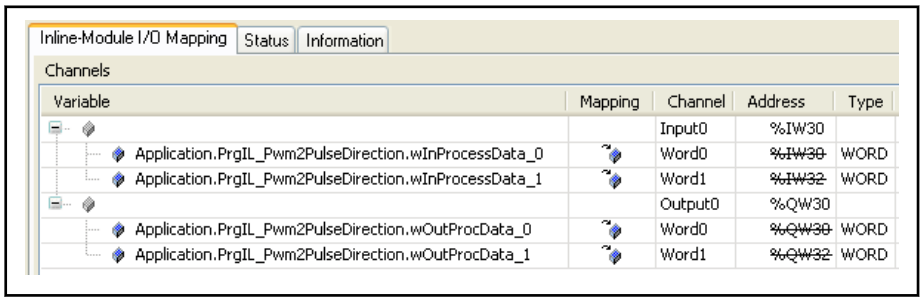


Fig. 14-46: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“Frequency” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“TargetPosition” is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response

Tab. 14-102: Error codes of the IL_Pwm2PulseDirection

14.4.8 IL_Pwm2SinglePulse

Brief description

The function block **IL_Pwm2SinglePulse** parameterizes the **R-IB IL PWM/2-PAC** terminal (Inline terminal for pulse width and frequency modulation) in the operation mode **Single Shot** (single pulse generator) to generate single pulses with a variable period.

One function block instance is required for each channel of the Inline I/O terminal. Both channels operate individually.



Except for the operation mode "Pulse direction signal", an own operation mode can be selected for each channel. If the terminal operates in the "Pulse direction signal" mode, both outputs are required.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

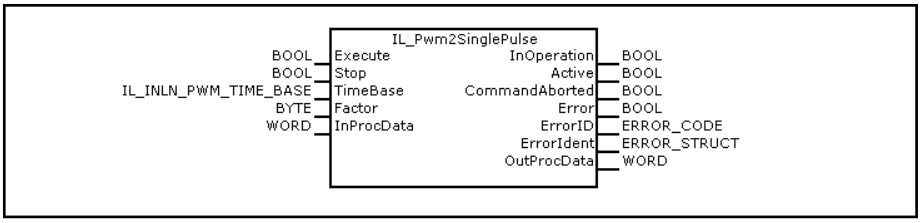


Fig. 14-47: IL_Pwm2SinglePulse function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	When the edge is positive, function block processing starts
	Stop	BOOL	Stops the pulse output
	TimeBase	IL_INLN_PWM_TIME_BASE	Time base, see "IL_INLN_PWM_TIME_BASE" on page 651
	Factor	BYTE	Factor, Pulse duration = time base x factor 0 stops the single shot function
	InProcData	WORD	Process input word
VAR OUTPUT	Done	BOOL	Computation completed
	Active	BOOL	Computation running (pulse output)
	CommandAborted	BOOL	Computation aborted by other function block or stop
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 652
	OutProcData	WORD	Process output word

Tab. 14-103: IL_Pwm2SinglePulse interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Stop	BOOL	FALSE	TRUE	FALSE	Continuous
TimeBase	ENUM	TIME_10US	TIME_100MS	TIME_10US	Positive edge at "Execute"
Factor	BYTE	0	255	0	Positive edge at "Execute"

Tab. 14-104: Input behavior of the IL_Pwm2SinglePulse

Functional description

Enable the function block instance only after the first PLC cycle by a positive edge at the **Execute** input.

The function block configures the channel as single pulse generator by specifying the time base and a factor.

In this operation mode, the channel outputs single pulses with a length between 10 µs and ... 25.5 s:

```
Pulse length = TimeBase * Factor
```

After applying this data, **Active** becomes TRUE and the pulse is generated at the terminal output.

A positive edge at the **Stop** input ends the pulse output and enables the **CommandAborted** output.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

At a positive edge at the **Execute** input, errors are deleted and the terminal is reconfigured.

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:

   Byte 0/1/2/3 = Plug 4/3/2/1.
2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

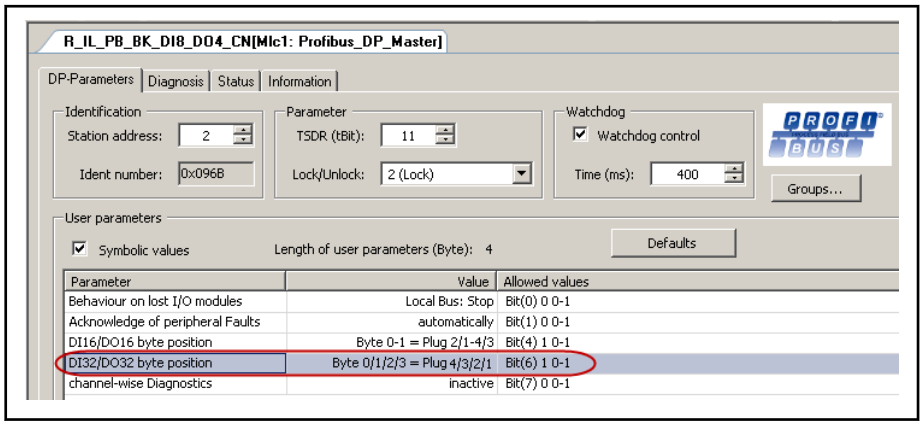


Fig. 14-48: Dialog: Bus terminal with changed byte sequence

IL_INLN_PWM_TIME_BASE

The enumeration specifies the time base of the single pulse generation.

ENUM	Description
TIME_10US	10 microseconds
TIME_100US	100 microseconds
TIME_1MS	1 millisecond
TIME_10MS	10 milliseconds
TIME_100MS	100 milliseconds

Example

The function block is used to configure a PWM terminal in the "Single pulse generation" mode.

The settings contain **TimeBase** and **Factor**. Process data has to be mapped.

In this example, the function block **Pwm2SinglePulse** is configured as single channel with a length of 1 s.

Declaration of the example

```
PROGRAM PrgIL_Pwm2SinglePulse
VAR
    fbIL_Pwm2SinglePulse: IL_Pwm2SinglePulse;
    wInProcessData : WORD;
    wOutProcessData: WORD;
    bDone: BOOL;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_Pwm2SinglePulse.TimeBase    := IL_INLN_PWM_TIME_BASE.TIME_10MS;
 fbIL_Pwm2SinglePulse.Stop        := FALSE;
 fbIL_Pwm2SinglePulse.Factor      := BYTE#100;
 fbIL_Pwm2SinglePulse.InProcData  := wInProcessData;
 wOutProcessData                  := fbIL_Pwm2SinglePulse.OutProcData;

 // FB call
  fbIL_Pwm2SinglePulse(Execute := TRUE);

 // Interrupting the function block
  IF fbIL_Pwm2SinglePulse.Active = TRUE THEN
      fbIL_Pwm2SinglePulse.Stop  := TRUE;
  END_IF

 // Confirmation of pulse execution completed
   bDone := fbIL_Pwm2SinglePulse.Done;
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData" and "wOutProcessData" are declared in the program "prgIL_Pwm2SinglePulse".

The first channel is mapped in the program.

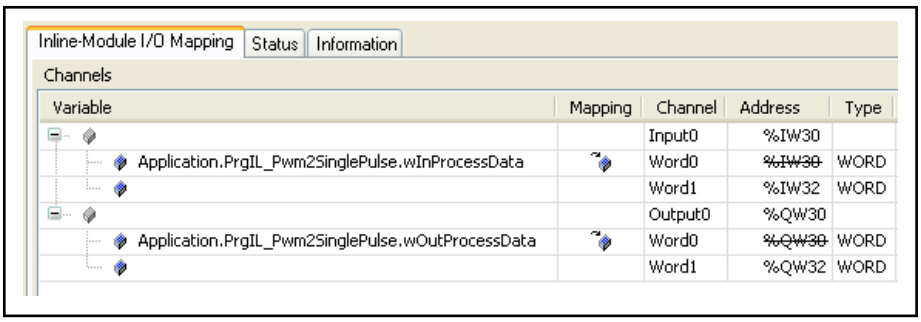


Fig. 14-49: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“Frequency” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“Factor” is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response

Tab. 14-105: Error codes of the IL_Pwm2SinglePulse

14.4.9 IL_Ssi

Brief description

The function block **IL_Ssi** configures the positioning terminal for the absolute value encoder **R-IB IL SSI-PAC**. Apart from the complete function logic to control the positioning, the positioning terminal also includes the actual position detection (input for absolute value encoder) and the drive control (four digital outputs). It can be used for the point-to-point positioning for rotary axes as well as for linear axes.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

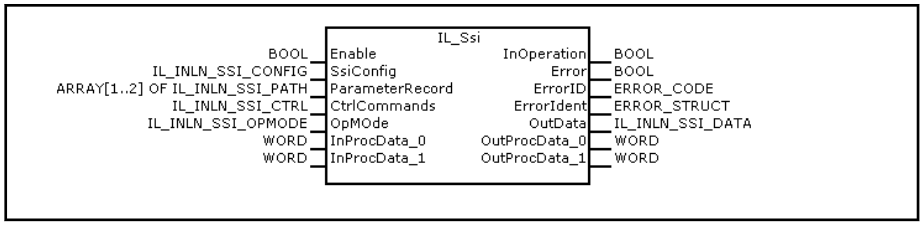


Fig. 14-50: IL_Ssi function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	SsiConfig	IL_INLN_SSI_CONFIG	Terminal configuration, see "IL_INLN_SSI_CONFIG" on page 655
	ParameterRecord	ARRAY[1..2] OF IL_INLN_SSI_PATH	Parameterization of the traversing path, see "IL_INLN_SSI_CONFIG" on page 655
	CtrlCommands	IL_INLN_SSI_CTRL	Command parameter to control the positioning terminal, see "IL_INLN_SSI_CTRL" on page 658
	OpMode	IL_INLN_SSI_OPMODE	Operation mode selection, see "IL_INLN_SSI_OPMODE" on page 660
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 665
	OutData	IL_INLN_SSI_DATA	Status of the positioning terminal, see "IL_INLN_SSI_DATA" on page 660
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-106: IL_Ssi interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
OpMode	IL_INLN_SSI_OPMODE	0	2	0	Continuously with "Enable"

Tab. 14-107: Input behavior of the IL_Ssi

Functional description

The function block configures the **R-IB IL SSI** positioning terminal, parameterizes traversing paths and controls the positioning.

1. **Configuration:** The structure "IL_INLN_SSI_CONFIG" on page 655 contains all input variables to initialize the terminal.

   At a positive edge at **Enable**, the terminal is initialized and the user specifications are configured accordingly.

After initialization, the function block sets **InOperation** to TRUE.

2. **Parameterization of the traversing path:** The structure "IL_INLN_SSI_PATH" on page 657 contains all data required to parameterize the traversing path. The positioning terminal allows to parameterize two independent traversing paths using two positioning data sets.

   The variable **ParameterRecord** contains two elements of the type IL_INLN_SSI_PATH and thus allows the individual parameterization of two traversing paths.

   A positive edge at **DefineParameterRecord** applies the data.

3. **Control position**: The positioning terminal supports three operation modes to control the position and to read the status.
   - **Control position and read position**: Positioning can be controlled and the current position can be simultaneously read in this operation mode
   - **Control position and read velocity**: Positioning can be controlled and the current velocity can be simultaneously read in this operation mode
   - **Control position and read status**: Positioning can be controlled and the current status can be simultaneously read in this operation mode

Use the **OpModeinput** to select the different operation modes "IL_INLN_SSI_OPMODE" on page 660. After selecting the operation mode, select the command parameters "IL_INLN_SSI_CTRL" on page 658 to control the positioning terminal.

The structure "IL_INLN_SSI_DATA" on page 660 contains all status and position information.

If an error occurs, the function block generates the resulting information and sends it to **Error**, **ErrorID** and **ErrorIdent**.

☞

1. Define the encoder offset **before** the compilation factor
2. Determine the ratio between numerator and denominator of the compilation factor due to the mechanics.

   The compilation factor has to be ≤ 1!
3. Specify all other values in the measuring unit of the compilation factor now:
   - Software limit switch
   - Parameter sets (startup window, quick startup window, pre-stop window, stop window, target window, target position)
   - Drive standstill
   - Logic offset

   The terminal also reports all values (e.g. actual position) in this measuring unit.

---

1. If the terminal is used with Profibus or Profinet field bus, change the byte sequence:
   Byte 0/1/2/3 = Plug 4/3/2/1.
2. Changing the byte sequence at the bus terminal affects all terminals downstream of the bus terminal!

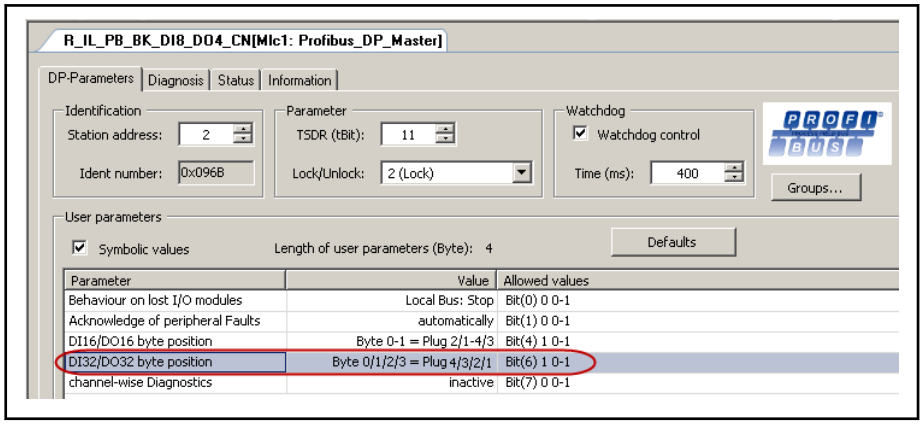


Fig. 14-51: Dialog: Bus terminal with changed byte sequence

IL_INLN_SSI_CONFIG

The structure IL_INLN_SSI_CONFIG contains all parameters to initialize the terminal.

Name	Type	Description	Min. value	Max. value	Default value	Effective
Parity	BYTE	Encoder parity 0 - None 1 – Even 2 - Odd	0	2	0	Positive edge at "Enable"
EncRotation	BOOL	Encoder installation direction FALSE – Normal, TRUE - Inverted	FALSE	TRUE	FALSE	Positive edge at "Enable"
Resolution	BYTE	Encoder resolution (8 .. 26)	8	26	8	Positive edge at "Enable"
GrayCode	BOOL	FALSE - Binary code TRUE - Gray code	FALSE	TRUE	FALSE	Positive edge at "Enable"
AxType	BYTE	Axis type (rotary axis/linear axis) 0 - Linear axis 1 - Rotary axis: Motion only possible in a negative direction of rotation 2 - Rotary axis: Motion only possible in a positive direction of rotation 3 – Rotary axis: Negative and positive hysteresis	0	3	0	Positive edge at "Enable"

Name	Type	Description	Min. value	Max. value	Default value	Effective
BusReset	BOOL	Response of the outputs at bus reset FALSE – At a bus reset, resets all switching outputs and stops a running positioning. TRUE - A bus reset does not affect the positioning, i.e. positionings can be started by the inputs.	FALSE	TRUE	FALSE	Positive edge at "Enable"
OutputFunctions	BYTE	Switching outputs (output variant) 0 – Invalid 1 – Version1 2 – Version2 3 – Version3 4 – Version4 5 – Version 5 For more detailed information, refer to the "Version details" data sheet	0	5	0	Positive edge at "Enable"
Initiator3Function	BYTE	IN3F: Function initiator 3 1 - 0/1 Start position with parameter record 2 2 - 0/1 Start position with parameter record 1 3 – None (digital input only)	1	3	1	Positive edge at "Enable"
Initiator2Function	BYTE	IN2F: Function initiator 2 0 - Maximum limit switch 1 - 1/0 Maximum limit switch 2 - 0/1 Start position with parameter record 2 3 – None (digital input only)	0	3	0	Positive edge at "Enable"
Initiator1Function	BYTE	IN1F: Function initiator 1 0 - Minimum limit switch 1 - 1/0 Minimum limit switch 2 - 0/1 Start position with parameter record 1 3 – None (digital input only)	0	3	0	Positive edge at "Enable"
DefineEncOffset	BOOL	Definition of encoder offset	FALSE	TRUE	FALSE	Continuously with "Enable"
EncOffset	DWORD	Encoder offset 0 = No encoder offset 1 ... (2^{26} - 1) increments	0	67108863	0	Positive edge at "DefineEncOffset"
DefineGearRatio	BOOL	Definition of transmission ratio	FALSE	TRUE	FALSE	Continuously with "Enable"

Name	Type	Description	Min. value	Max. value	Default value	Effective
Numerator-GearRatio	WORD	Gear ratio of numerator, e.g. traversing distance in cm (Numerator < denominator)	1	1023	1	Positive edge at "DefineGearRatio"
Denominator-GearRatio	WORD	Gear ratio denominator i.e. traversing distance in increments	1	65535	1	Positive edge at "DefineGearRatio"
DefineLogicOffset	BOOL	Definition of logic offset Offset of the position values	FALSE	TRUE	FALSE	Continuously with "Enable"
LogicOffset	DINT	Logic offset	-33554432	33554431	0	Positive edge at "DefineLogicOffset"
DefineMinMaxSWLimSwitch	BOOL	Definition of the software limit switch Maximum software limit switch, minimum software limit switch	FALSE	TRUE	FALSE	Continuously with "Enable"
MinSWLimSwitch	DINT	Minimum software limit switch	-33554432	33554431	0	Positive edge at "DefineMinMaxSWLimSwitch"
MaxSWLimSwitch	DINT	Maximum software limit switch	-33554432	33554431	0	Positive edge at "DefineMinMaxSWLimSwitch"
DefineDsdOd	BOOL	Definition of waiting time for direction and standstill detection and output delay time	FALSE	TRUE	FALSE	Continuously with "Enable"
DrvStartingDelay	BYTE	Waiting time for direction and standstill detection	0	255	0	Positive edge at "DefineDsdOd"
OutDelayTime	WORD	Output delay time	0	4095	0	Positive edge at "DefineDsdOd"
DefineJogMode	BOOL	Definition of jog mode	FALSE	TRUE	FALSE	Continuously with "Enable"
JogDelayTimeCount	BYTE	Delay time for the permanent signal at jog mode	0	255	0	Positive edge at "DefineJogMode"
JogMonoflopTime	WORD	Monoflop time for jog mode	0	65535	0	Positive edge at "DefineJogMode"
DefineDrvStop	BOOL	Definition of drive standstill	FALSE	TRUE	FALSE	Continuously with "Enable"
DrvStopTime	BYTE	Time interval, drive standstill	0	31	0	Positive edge at "DefineDrvStop"
DrvStopDist	WORD	Distance, drive standstill	0	65535	0	Positive edge at "DefineDrvStop"

IL_INLN_SSI_PATH The positioning terminal allows to parameterize two independent traversing paths using two positioning data sets. This structure contains all parameters required to configure the traversing path.

Name	Type	Description	Min. value	Max. value	Default value	Effective
DefineParameterRecord	BOOL	Positive edge writes values required to parameterize the traversing distance	FALSE	TRUE	FALSE	Continuously with "Enable"
DefineTargetPos	BOOL	Positive edge defines target position. A positive edge at DefineParameterRecord has the same effect	FALSE	TRUE	FALSE	Continuously with "Enable"
StartWindow	DWORD	"Start window" of the traversing distance 0 ... (2^{26} - 1) increments	0	67108863	0	Positive edge at "DefineParameterRecord"
RapidStartWindow	DWORD	"Rapid start window" of the traversing distance 0 ... (2^{26} - 1) increments	0	67108863	0	Positive edge at "DefineParameterRecord"
RapidStopWindow	DWORD	"Rapid stop window" of the traversing distance 0 ... (2^{26} - 1) increments	0	67108863	0	Positive edge at "DefineParameterRecord"
PreStopWindow	DWORD	"Pre-stop window" of the traversing distance 0 ... (2^{26} - 1) increments	0	67108863	0	Positive edge at "DefineParameterRecord"
StopWindow	DWORD	"Stop window" of the traversing distance 0 ... (2^{26} - 1) increments	0	67108863	0	Positive edge at "DefineParameterRecord"
RepeatCounter	BYTE	Target repeating counter 0 ... 15 repetitions	0	15	0	Positive edge at "DefineParameterRecord"
TargetWindow	WORD	Target window (0 ... 2^{16} - 1) increments	0	65535	0	Positive edge at "DefineParameterRecord"
TargetPos	DINT	Target position -2^{25} ... +(2^{25} - 1) increments	-33554432	33554431	0	Positive edge at "DefineParameterRecord/ DefineTargetPos"
LubFricCompValue	DINT	Lubrication and friction compensation value -2^{25} ... +(2^{25} - 1) increments	-33554432	33554431	0	Positive edge at "DefineParameterRecord"

IL_INLN_SSI_CTRL The structure contains the command parameters to control the positioning terminal.

Name	Type	Description	Min. value	Max. value	Default value	Effective
CtrlOut4	BOOL	Output 4 If output version 5 has been activated, otherwise not relevant	FALSE	TRUE	FALSE	Continuously with "Enable"
CtrlOut3	BOOL	Output 3 If output version 5 has been activated, otherwise not relevant	FALSE	TRUE	FALSE	Continuously with "Enable"
CtrlOut2	BOOL	Output 2 If output version 5 has been activated, otherwise not relevant	FALSE	TRUE	FALSE	Continuously with "Enable"
CtrlOut1	BOOL	Output 1 If output version 5 has been activated, otherwise not relevant	FALSE	TRUE	FALSE	Continuously with "Enable"
JogNegative	BOOL	Jog mode in negative direction	FALSE	TRUE	FALSE	Continuously with "Enable"
JogPositive	BOOL	Jog mode in positive direction	FALSE	TRUE	FALSE	Continuously with "Enable"
EnableLubFricComp	BOOL	Enable lubrication and friction compensation	FALSE	TRUE	FALSE	Continuously with "Enable"
EnableLooping	BOOL	Enable looping	FALSE	TRUE	FALSE	Continuously with "Enable"
DefDrvDir	BOOL	Define approaching direction of target position during active backlash compensation or looping 0: Positive direction 1: Negative direction during active backlash compensation or looping	FALSE	TRUE	FALSE	Continuously with "Enable"
EnableBacklashComp	BOOL	Activate backlash compensation The direction is set in DefDrvDir	FALSE	TRUE	FALSE	Continuously with "Enable"
EnableLocalCtrl2	BOOL	Enable local positioning control with parameter set 2 using digital inputs (initiators)	FALSE	TRUE	FALSE	Continuously with "Enable"
StartPosDrv2	BOOL	0/1 Start position with parameter set 2	FALSE	TRUE	FALSE	Continuously with "Enable"
EnableLocalCtrl1	BOOL	Enable local positioning control with parameter set 1 using digital inputs (initiators)	FALSE	TRUE	FALSE	Continuously with "Enable"
StartPosDrv1	BOOL	0/1 Start position with parameter set 1	FALSE	TRUE	FALSE	Continuously with "Enable"
Stop	BOOL	Positioning is stopped immediately. The currently executed command is aborted. It can only be positioned again after resetting the stop bit.	FALSE	TRUE	FALSE	Continuously with "Enable"

IL_INLN_SSI_DATA The structure contains the status parameters of the positioning terminal.

Name	Type	Description
ActPos	DINT	Actual position
Speed	DINT	Actual velocity
StatusOfOut4	BOOL	Status of the output 4
StatusOfOut3	BOOL	Status of the output 3
StatusOfOut2	BOOL	Status of the output 2
StatusOfOut1	BOOL	Status of the output 1
StatusOfIn4	BOOL	Status of the input 4
StatusOfIn3	BOOL	Status of the input 3
StatusOfIn2	BOOL	Status of the input 2
StatusOfIn1	BOOL	Status of the input 1
PosP2InProgress	BOOL	Positioning with parameter set 2 is being processed
PosP2Complete	BOOL	Positioning with parameter set 2 was completed successfully
PosP1InProgress	BOOL	Positioning with parameter set 1 is being processed
PosP1Complete	BOOL	Positioning with parameter set 1 was completed successfully
InitNotComplete	BOOL	Terminal not yet completely initialized
DrvStop	BOOL	Drive standstill was detected in the monitoring of the drive standstill is disabled.
ParameterizationInProgress	BOOL	TRUE - Parameterization is being processed

IL_INLN_SSI_OPMODE The enumeration supports the operation mode selection.

ENUM	Description
CTRL_POS_READ_POS	Control position and read position
CTRL_POS_READ_VEL	Control position and read velocity
CTRL_POS_READ_STATUS	Control position and read status

Example For the positioning, the positioning control uses the rapid traverse/creep mode.

Two drives are moved with two specified velocities (first rapid traverse, then creep mode) until reaching the switch-off point. The drive is then switched off. That means, positioning takes place in rapid traverse/creep mode. The positioning control switches on the output signals to control the drive after the positioning start and switches when the axis exceeds the predefined position threshold. It is first switched from rapid traverse to creep mode and then from creep mode to stop.

If the target position was reached, the drive is switched off. Thus, the drive does not provide any torque in the target position.

Essential steps to configure and operate the positioning terminal:

1. Configure encoder (mandatory) – Parity, resolution, code, direction of rotation
2. Configure axis type (mandatory) – Linear or rotary, initiator functions 1, 2 and 3, output functions

3. Configure encoder offset – Configure encoder offset before compilation factor
4. Define compilation factor – Compilation factor should be <= 1
5. Define logic offset – Position value offset. Measuring unit for all values same as for compilation factor
6. Define software limit switch - Measuring unit for all values same as for compilation factor
7. Define drive standstill – Maximum distance and time up to drive standstill
8. Define waiting time for direction and standstill detection
9. Define positioning data sets for two individual traversing paths
10. Control position and read position/velocity status – Operation mode selection
11. Select command parameters to control the positioning terminal to perform action



Only one job can be executed at a time. Thus, follow the list step by step.

E.g. Configure encoder offset, define compilation factor, define software limit switch, etc.

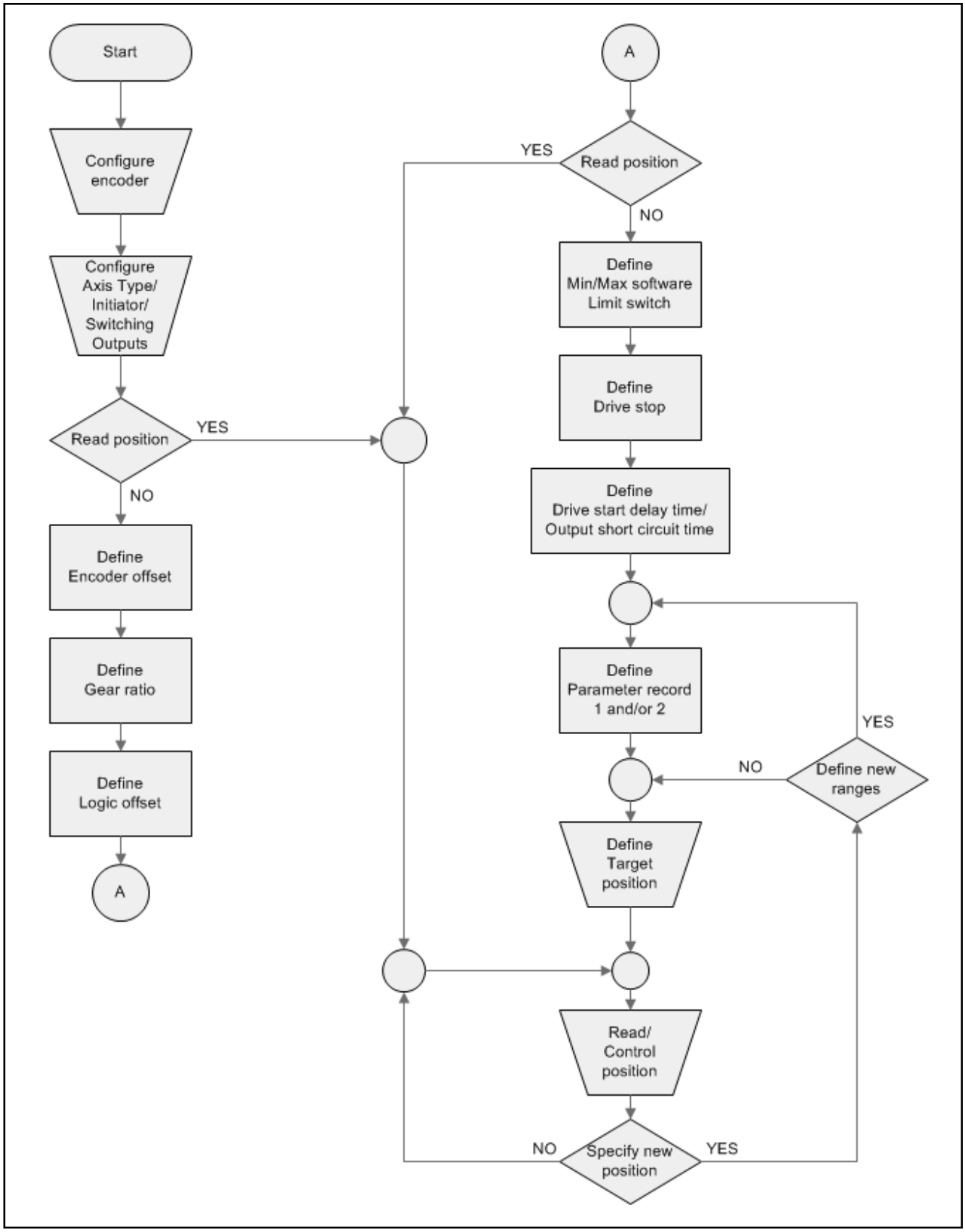


Fig. 14-52: Overview on command sequence

Declaration of the example

```
PROGRAM PrgIL_Ssi
VAR
    fbIL_Ssi: IL_Ssi;
    diActualPos: DINT;
    bDefineEncOffset: BOOL;
    RTrig_GearRatio: R_TRIG;
    bDefineGearRatio: BOOL;
    bDefineLogicOffset: BOOL;
    bDefineSWLimSwitch: BOOL;
    bDefineDsdOd: BOOL;
    bDefineDrvStop: BOOL;
    bDefineJogMode: BOOL;
    bDefineParameterRecord1: BOOL;
    bDefineParameterRecord2: BOOL;
    bStartPosDrv1: BOOL;
    bStartPosDrv2: BOOL;
    bJogPositive: BOOL;
    bJogNegative: BOOL;
    bStop: BOOL;
    bEnableLooping: BOOL;
    bEnableLocalCtrl1: BOOL;
    bEnableLocalCtrl2: BOOL;
    bEnableBacklashComp: BOOL;
    dwEncOffset: DWORD;
    bEnableLubFricComp: BOOL;
    bPosP1Complete: BOOL;
    bPosP2Complete: BOOL;
END_VAR
```

Implementation of the example

Minimum configuration to read position

```
fbIL_Ssi.SsiConfig.Parity := BYTE#0;
fbIL_Ssi.SsiConfig.Resolution:= BYTE#12;
fbIL_Ssi.SsiConfig.GrayCode:= TRUE;
fbIL_Ssi.SsiConfig.EncRotation:= FALSE;
fbIL_Ssi.SsiConfig.AxType:= BYTE#3;
fbIL_Ssi.SsiConfig.Initiator1Function:= BYTE#0;
fbIL_Ssi.SsiConfig.Initiator2Function:= BYTE#0;
fbIL_Ssi.SsiConfig.Initiator3Function:= BYTE#2;
fbIL_Ssi.SsiConfig.OutputFunctions:= BYTE#1;
fbIL_Ssi.InProcData_0:= InputProcessData_0;
fbIL_Ssi.InProcData_1:= InputProcessData_1;
OutputProcessData_0:= fbIL_Ssi.OutProcData_0;
OutputProcessData_1:= fbIL_Ssi.OutProcData_1 ;

fbIL_Ssi(Enable:=TRUE);

IF    fbIL_Ssi.InOperation = TRUE THEN
    fbIL_Ssi.OpMode:= IL_INLN_SSI_OPMODE.CTRL_POS_READ_POS;
    diActualPos := fbIL_Ssi.OutData.ActPos;
END_IF
```

Configure encoder offset. 0/1 at bDefineEncOffset sets encoder offset

```
IF    fbIL_Ssi.InOperation = TRUE THEN
    fbIL_Ssi.SsiConfig.EncOffset:= dwEncOffset;
    fbIL_Ssi.SsiConfig.DefineEncOffset:= bDefineEncOffset;
END_IF
```

Parameterize terminal. Positive edge at the respective control bit to apply data

```
IF    fbIL_Ssi.InOperation = TRUE THEN
    fbIL_Ssi.SsiConfig.NumeratorGearRatio:= WORD#1000;
    fbIL_Ssi.SsiConfig.DenominatorGearRatio:= WORD#4096;
    fbIL_Ssi.SsiConfig.MinSWLimSwitch:= DINT#-200;
    fbIL_Ssi.SsiConfig.MaxSWLimSwitch:= DINT#33445531;
    fbIL_Ssi.SsiConfig.DrvStartingDelay:= BYTE#50;
    fbIL_Ssi.SsiConfig.OutDelayTime:= WORD#10;
    fbIL_Ssi.SsiConfig.DrvStopDist:= WORD#200;
    fbIL_Ssi.SsiConfig.DrvStopTime:= BYTE#10;
```

```
    fbIL_Ssi.SsiConfig.LogicOffset:= DINT#0;
    fbIL_Ssi.SsiConfig.JogDelayTimeCount:= BYTE#20;
    fbIL_Ssi.SsiConfig.JogMonoflopTime:= WORD#10;

    fbIL_Ssi.SsiConfig.DefineGearRatio := bDefineGearratio;
    fbIL_Ssi.SsiConfig.DefineLogicOffset:= bDefineLogicOffset;
    fbIL_Ssi.SsiConfig.DefineMinMaxSWLimSwitch:= bDefineSWLimSwitch;
    fbIL_Ssi.SsiConfig.DefineDsdOd:= bDefineDsdOd;
    fbIL_Ssi.SsiConfig.DefineDrvStop:= bDefineDrvStop;
    fbIL_Ssi.SsiConfig.DefineJogMode:= bDefineJogMode;

END_IF
```

Parameterizing positioning data sets. Positive edge at the respective definition input to apply data

```
IF    fbIL_Ssi.InOperation = TRUE THEN

    fbIL_Ssi.ParameterRecord[1].StartWindow:= DWORD#10;
    fbIL_Ssi.ParameterRecord[1].RapidStartWindow:= DWORD#250;
    fbIL_Ssi.ParameterRecord[1].RapidStopWindow:= DWORD#20;
    fbIL_Ssi.ParameterRecord[1].PreStopWindow:= DWORD#10;
    fbIL_Ssi.ParameterRecord[1].StopWindow:= DWORD#5;
    fbIL_Ssi.ParameterRecord[1].LubFricCompValue:= 0;
    fbIL_Ssi.ParameterRecord[1].TargetWindow:= WORD#2;
    fbIL_Ssi.ParameterRecord[1].TargetPos:=DINT#550;

    fbIL_Ssi.ParameterRecord[1].DefineParameterRecord:=bDefineParameterRecord1;

    fbIL_Ssi.ParameterRecord[2].StartWindow:= DWORD#10;
    fbIL_Ssi.ParameterRecord[2].RapidStartWindow:= DWORD#200;
    fbIL_Ssi.ParameterRecord[2].RapidStopWindow:= DWORD#20;
    fbIL_Ssi.ParameterRecord[2].PreStopWindow:= DWORD#10;
    fbIL_Ssi.ParameterRecord[2].StopWindow:= DWORD#5;
    fbIL_Ssi.ParameterRecord[2].LubFricCompValue:= 0;
    fbIL_Ssi.ParameterRecord[2].TargetWindow:= WORD#2;
    fbIL_Ssi.ParameterRecord[2].TargetPos:=DINT#245;

    fbIL_Ssi.ParameterRecord[2].DefineParameterRecord:=bDefineParameterRecord2;
END_IF
```

control commands. Set the corresponding control bit to control the positioning terminal. Switch to operation mode "Control position and read status" during positioning.

```
IF    fbIL_Ssi.InOperation = TRUE THEN
    fbIL_Ssi.OpMode:= IL_INLN_SSI_OPMODE.CTRL_POS_READ_STATUS;
    fbIL_Ssi.CtrlCommands.StartPosDrv1:= bStartPosDrv1;
    fbIL_Ssi.CtrlCommands.StartPosDrv2:= bStartPosDrv2;
    fbIL_Ssi.CtrlCommands.JogPositive:= bJogPositive;
    fbIL_Ssi.CtrlCommands.JogNegative:= bJogNegative;
    fbIL_Ssi.CtrlCommands.Stop:= bStop;
    fbIL_Ssi.CtrlCommands.EnableLooping:= bEnableLooping;
    fbIL_Ssi.CtrlCommands.EnableLocalCtrl1:= bEnableLocalCtrl1;
    fbIL_Ssi.CtrlCommands.EnableLocalCtrl2:= bEnableLocalCtrl2;
    fbIL_Ssi.CtrlCommands.EnableBacklashComp:= bEnableBacklashComp;
    fbIL_Ssi.CtrlCommands.EnableLubFricComp:= bEnableLubFricComp;

END_IF
```

Read status

```
IF    fbIL_Ssi.OpMode = IL_INLN_SSI_OPMODE.CTRL_POS_READ_STATUS THEN
    bPosP1Complete:= fbIL_Ssi.OutData.PosP1Complete;
    bPosP2Complete:= fbIL_Ssi.OutData.PosP2Complete;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_Ssi".

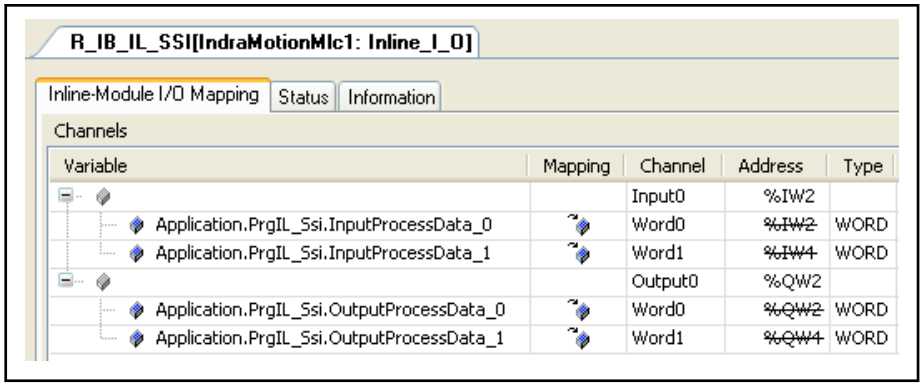


Fig. 14-53: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	Parity is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	Encoder resolution is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000003	Axis type is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000004	Output function is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000005	Initiator 3 function is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000006	Initiator 2 function is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000007	Initiator 1 function is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000008	Drive stop time is not valid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000009	Numerator gear ratio is not valid or Numerator gear ratio is greater than denominator gear ratio
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000A	Denominator gear ratio is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000B	Logical offset is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000C	Minimum software limit switch is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000D	Maximum software limit switch is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000E	Output delay time in invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000F	"Start window" of the traversing distance parameter set 1 is invalid

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000010	"Rapid start window" of the traversing distance parameter set 1 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000011	"Rapid stop window" of the traversing distance parameter set 1 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000012	"Pre-stop window" of the traversing distance parameter set 1 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000013	"Stop window" of the traversing distance parameter set 1 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000014	Target traveling repetition counter parameter set 1 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000015	Target position parameter set 1 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000016	Lubrication and friction compensation value parameter set 1 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000017	"Start window" of the traversing distance parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000018	"Rapid start window" of the traversing distance parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000019	"Rapid stop window" of the traversing distance parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000001A	"Pre-stop window" of the traversing distance parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000001B	"Stop window" of the traversing distance parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000001C	Target traveling repetition counter parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000001D	Target position parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000001E	Lubrication and friction compensation value parameter set 2 is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000001F	Encoder offset is invalid or Encoder offset is greater than the encoder resolution
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#00003020	16#00000001	Terminal is in HHOP mode. Control via the bus is not possible in this mode
DEVICE_ERROR (16#0008)	16#00003020	16#00000002	Output driver overloaded or short circuit
DEVICE_ERROR (16#0008)	16#00003020	16#00000003	Encoder supply fault. No voltage supply or short circuit
DEVICE_ERROR (16#0008)	16#00003020	16#00000004	Encoder configuration is valid
DEVICE_ERROR (16#0008)	16#00003020	16#00000005	Parity error Connection to the sensor is defective or sensor configured incorrectly
DEVICE_ERROR (16#0008)	16#00003020	16#00000006	Initiator or switching output configuration is invalid

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR (16#0008)	16#00003020	16#00000007	Reserved
DEVICE_ERROR (16#0008)	16#00003020	16#00000008	Control command is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000009	Target position outside the permitted range or it is not defined
DEVICE_ERROR (16#0008)	16#00003020	16#0000000A	Function cannot be executed, as software limit switches would crossed
DEVICE_ERROR (16#0008)	16#00003020	16#0000000B	Motor is not moving or is moving to the wrong direction
DEVICE_ERROR (16#0008)	16#00003020	16#0000000C	Reserved
DEVICE_ERROR (16#0008)	16#00003020	16#0000000D	The distance to the target position is shorter than the sum of the stop and start windows
DEVICE_ERROR (16#0008)	16#00003020	16#0000000E	Drive stop detected
DEVICE_ERROR (16#0008)	16#00003020	16#0000000F	Software end limit switch reached
DEVICE_ERROR (16#0008)	16#00003020	16#00000010	Hardware end limit switch reached
DEVICE_ERROR (16#0008)	16#00003020	16#00000011	The target window could not be reached
DEVICE_ERROR (16#0008)	16#00003020	16#00000012	Counter overflow
DEVICE_ERROR (16#0008)	16#00003020	16#00000013	Invalid action
DEVICE_ERROR (16#0008)	16#00003020	16#00000014	Bus reset
DEVICE_ERROR (16#0008)	16#00003020	16#00000015	Invalid system configuration

Tab. 14-108: Error codes of the IL_Ssi

14.4.10 IL_SsiIn

Brief description

The function block **IL_SsiIn** configures the Inline terminal **R-IB IL SSI IN-PAC** for absolute value encoders with SSI interface and retrieves data of up to 25 bits from the absolute value encoder with SSI interface (single-turn, multi-turn or length measuring systems) with gray and binary code and converts gray code into binary code.

One function block instance is required for each Inline I/O terminal.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

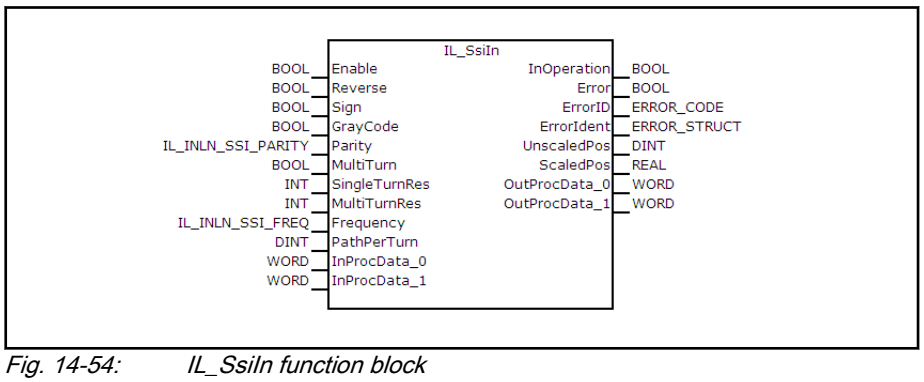


Fig. 14-54: IL_SsiIn function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	Reverse	BOOL	Direction of rotation reversal FALSE = Off TRUE = On
	Sign	BOOL	FALSE = Encoder value without sign bit TRUE = Encoder value with sign bit (MSB)
	GrayCode	BOOL	Parameter indicates the coding used by the encoder to provide information. FALSE = Terminal forwards data unchanged to local bus. TRUE = Terminal converts gray-coded data into binary-coded data and forwards it to the local bus.
	Parity	IL_INLN_SSI_PARITY	Parity, see "IL_INLN_SSI_PARITY" on page 669
	MultiTurn	BOOL	TRUE = Multi turn FALSE = Single turn
	SingleTurnRes	INT	Resolution of single turn: 8 to 25 bits Note: (SingleTurnRes + MultiTurnRes) <= 25
	MultiTurnRes	INT	Resolution of multi turn: 0 to 25 bits Note: (SingleTurnRes + MultiTurnRes) <= 25
	Frequency	IL_INLN_SSI_FREQ	Transmission rate used to read encoder data via SSI interface, see "IL_INLN_SSI_FREQ" on page 670
	PathPerTurn	DINT	Distance covered in one encoder revolution
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 671
	UnScaledPos	DINT	Unscaled position value (original value)
	ScaledPos	REAL	Scaled position value
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-109: IL_SsiIn interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
Reverse	BOOL	FALSE	TRUE	FALSE	Positive edge at "Enable"
Sign	BOOL	FALSE	TRUE	FALSE	Continuously with Enable
GrayCode	BOOL	FALSE	TRUE	FALSE	Positive edge at "Enable"
Parity	IL_INLN_SSI_PARITY	None	Even	None	Positive edge at "Enable"
MultiTurn	BOOL	FALSE	TRUE	FALSE	Positive edge at "Enable"
SingleTurnRes	INT	8	25	0	Positive edge at "Enable"
MultiTurnRes	INT	0	25	0	Positive edge at "Enable"
Frequency	IL_INLN_SSI_FREQ	FREQ_100KHZ	FREQ_1MHZ	FREQ_100KHZ	Positive edge at "Enable"
PathPerTurn	DINT	0	65535	0	Positive edge at "Enable"

Tab. 14-110: Input behavior of the IL_SsiIn

Functional description

The function block is used to retrieve data of up to 25 bits from absolute value encoders with SSI interface (single turn, multi turn or length measuring systems). It supports encoders with gray and binary code and converts gray code into binary code.

The function blocks supports the following characteristics:

- (Single turn, multi turn or length measuring systems)
- 8 ... 25 bits (configurable)
- Gray code, binary code
- None, even or uneven parity monitoring
- Transmission rates of 100 kHz, 200 kHz, 400 kHz, 800 kHz, 1 MHz
- Direction of rotation reversal
- Assigning a sign bit (if supported by encoder)
- Offset default

A positive edge at **Enable** assigns the input parameters and enables the function block.

A reconfiguration is only initiated with a new positive edge at the **Enable** input.

If the input parameters are invalid, the function block generates an error and the resulting information is sent to **Error**, **ErrorID** and **ErrorIdent**.

An error can be cleared with a new positive edge at the **Enable** input.

IL_INLN_SSI_PARITY

Use this parameter to select the parity monitoring function for the SSI interface.

ENUM	Parity
NONE	None
ODD	Odd parity
EVEN	Even parity

IL_INLN_SSI_FREQ The transmission rate used to retrieve encoder data via the SSI interface is selected via this enumeration type.

ENUM	Transfer rate
FREQ_100KHZ	100 kHz
FREQ_200KHZ	200 kHz
FREQ_400KHZ	400 kHz
FREQ_800KHZ	800 kHz
FREQ_1MHZ	1 MHz

Example The function block reads data of up to 25 bits of an absolute encoder with SSI interface (single turn, multi turn or length measuring systems) with gray and binary code and converts gray code into binary code.

In this example, the absolute encoder is configured with a single-turn resolution of 8 bits, a multi-term resolution of 12 bits, no parity, gray code and 100 kHz transmission rate. **PathPerTurn** is the distance covered by the encoder in case of an encoder resolution and 100 is assumed as in the example. The original value read by the decoder is displayed in **UnScaledPos** and the scaled value is displayed in **ScaledPos**.

If the direction of rotation reversal is switched on, the terminal inverts all position values loaded by the encoder (the position value changes by activating this function).

The **Sign** bit can be used to invert the data read by the encoder.

Declaration of the example

```
PROGRAM PrgIL_SsiIn
VAR
    fbIL_SsiIn: IL_SsiIn;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    diUnScaledPos: DINT;
    rScaledPos: REAL;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_SsiIn.Frequency   := IL_INLN_SSI_FREQ.FREQ_100KHZ;
 fbIL_SsiIn.GrayCode    := TRUE;
 fbIL_SsiIn.Parity      := IL_INLN_SSI_PARITY.NONE;
 fbIL_SsiIn.SingleTurnRes := INT#8;
 fbIL_SsiIn.MultiTurn   := TRUE;
 fbIL_SsiIn.MultiTurnRes := INT#12;
 fbIL_SsiIn.Sign         := FALSE;
 fbIL_SsiIn.Reverse      := FALSE;
 fbIL_SsiIn.InProcData_0 := wInProcessData_0;
 fbIL_SsiIn.InProcData_1 := wInProcessData_1;
 wOutProcessData_0       := fbIL_SsiIn.OutProcData_0;
 wOutProcessData_1       := fbIL_SsiIn.OutProcData_1;
 fbIL_SsiIn.PathPerTurn  := DINT#100;

 // FB call
```

```
fbIL_SsiIn(Enable := TRUE);

// Data read
IF fbIL_SsiIn.InOperation = TRUE THEN
    diUnScaledPos    := fbIL_SsiIn.UnScaledPos;
    rScaledPos       := fbIL_SsiIn.ScaledPos;
END_IF

// For inverting the incoming data
IF  fbIL_SsiIn.InOperation = TRUE THEN
    fbIL_SsiIn.Sign         := TRUE;
END_IF

// For reversing the encoder pulses
  fbIL_SsiIn.Reverse    := TRUE;
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_SsiIn".

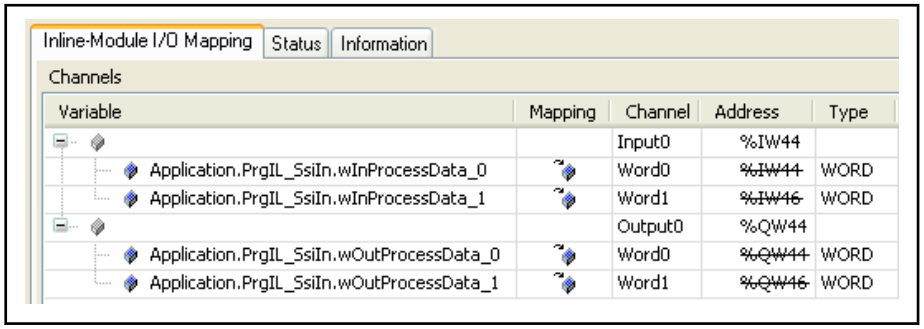


Fig. 14-55: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“Parity” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“SingleTurnRes” is invalid.
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000003	“MultiTurnRes” is invalid. (0< MultiTurnRes >25)
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000004	Parameter (SingleTurnRes + MultiTurnRes) > 25
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000005	“Frequency” is invalid.
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000006	“PathPerTurn” is invalid
DEVICE_ERROR (16#0008)	16#00003020	16#00000000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#00003020	16#00000042	No encoder supply present or short-circuited
DEVICE_ERROR (16#0008)	16#00003020	16#00000044	A parity error is a transmission error that occurred during the serial data transmission from the absolute encoder to the terminal

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR (16#0008)	16#00003020	16#00000048	Invalid configuration
DEVICE_ERROR (16#0008)	16#00003020	16#00000050	Unknown control code; an unknown control code was transferred

Tab. 14-111: Error codes of the IL_Ssi

14.4.11 IL_IncIn

Brief description

The function block **IL_IncIn** configures the Inline terminal **R-IB IL INC-IN-PAC** (Inline terminal for incremental value encoder) and normalizes its position values.

The function block converts all required parameters and measuring results into measuring units specified by the user.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos and Profinet IO field bus.

Interface description

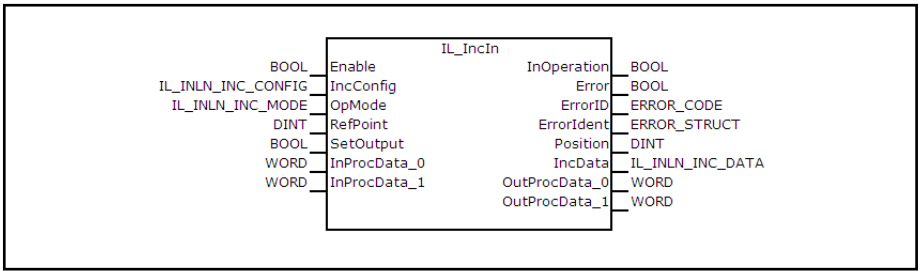


Fig. 14-56: IL_IncIn function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	IncConfig	IL_INLN_INC_CONFIG	Configuration data, see "IL_INLN_INC_CONFIG" on page 673
	OpMode	IL_INLN_INC_MODE	Operation modes, see "IL_INLN_INC_MODE" on page 674
	RefPoint	DINT	Reference point. For homing, this value is applied as initial value
	SetOutput	BOOL	This input controls the digital output of the terminal
	InProcData_0	WORD	Process input word 0
	InProcData_1	WORD	Process input word 1

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	If TRUE, the function block operates at its current task
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information, see "Troubleshooting" on page 676
	Position	DINT	Scaled position in "ReadPos" mode
	IncData	IL_INLN_INC_DATA	Status_I1/ Status_I2 / Status_I3/ IncValue and DeviceHomed, see "IL_INLN_INC_DATA" on page 674
	OutProcData_0	WORD	Process output value 0
	OutProcData_1	WORD	Process output value 1

Tab. 14-112: IL_IncIn interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
RefPoint	DINT	0	-------	0	Continuous
SetOutput	BOOL	FALSE	TRUE	FALSE	Continuous

Tab. 14-113: Input behavior of the IL_IncIn

Functional description

The function block is familiar with two operation modes:

1. **ReadPos:** to read and scale the position values of the terminal.
2. **Reference:** Supports the encoder homing. Up to six types of homing are possible.

The **ReadPos** mode starts if selected for **OpMode** and if the **Enable** input is enabled (positive edge).

With a positive edge, the **IncConfig** parameter (IL_INLN_INC_CONFIG structure) is applied automatically. Changes in parameter values do not affect the function of the terminal anymore. If values are changed, there has to be a positive edge at **Enable** again.

The 0/1 transition to **InOperation** indicates that the function block operates. Valid and scaled measuring values are output at **Position**. Additional status information on the terminal is available in the structured variable **IncData**.

The **Reference** mode starts if the function block was enabled at the **OpMode** input via one of the six homing types. During homing, the position values are invalid. If homing the terminal was successful, it is displayed by a 0/1 transition to **IncData.DeviceHomed**.

If an error occurs in one of the two operation mode, it can be identified and eliminated via **Error**, **ErrorID** and **ErrorIdent**.

IL_INLN_INC_CONFIG

The structure contains parameters to configure the Inline terminal **R-IB IL INC-IN**.

Name	Type	Description	Min. value	Max. value	Default value	Effective
EncoderType	BOOL	Connected encoder type TRUE: Asymmetric encoder FALSE: Symmetric encoder	FALSE	TRUE	FALSE	Positive edge at "Enable"
PulseFactor	INT	Possible pulse factors for encoder pulses 1 - single fold evaluation 2 - double fold evaluation 4 - four fold evaluation (Should be >0)	0	4	0	Positive edge at "Enable"
IncPerTurn	INT	Increments per encoder rotation (for encoder resolution, see encoder data sheet) (Should be >0)	0	65535	0	Positive edge at "Enable"
PathPerTurn	DINT	Distance covered by the machine per encoder revolution. This distance should be specified in the smallest measuring unit and has to be less than (IncPerTurn * PulseFactor). (Should be >0)	0	------	0	Positive edge at "Enable"
Offset	DINT	Offset of position value The value can be used to shift the counting range	0	------	0	Continuous

Tab. 14-114: Permitted values of IL_INLN_INC_CONFIG for IL_IncIn

IL_INLN_INC_DATA The structure contains output parameters of the terminal **R-IB IL INC-IN**.

Name	Type	Description
Status_I1	BOOL	Status input channel 1
Status_I2	BOOL	Status input channel 2
Status_I3	BOOL	Status input channel 3
DeviceHomed	BOOL	Reference point reached
IncValue	REAL	Content of the position counter (not scaled).

IL_INLN_INC_MODE The enumeration supports the selection of the operating state of the terminal.

ENUM	Description
READPOS	Reading actual position value
SETTO_REFERENCE	Set counter to reference point value (homing in standstill)
HOME_RISE_DI	Homing to the next positive edge of input I1, I2 or I3 without Z
HOME_FALL_DI	Homing to the next negative edge of input I1, I2 or I3 without Z
HOME_Z_RISE_DI	Homing to Z after positive edge of input I1, I2 or I3

ENUM	Description
HOME_Z_FALL_DI	Homing to Z after negative edge of input I1, I2 or I3
HOME_DELTA	Homing of distance-encoded incremental encoders
TESTMODE	Test mode

Example

This function block configures an Inline terminal for incremental value encoders to measure distances or to determine positions.

The settings include the input configuration and the operation mode selection. Process data has to be mapped.

The example shows the configuration of the **R-IB IL INC-IN-PAC** terminal for a simple evaluation and asymmetric encoder signal type.

ReadPos or **Homing** can be specified by selecting the operation mode in **OpMode**.

The function block reads the position in the **ReadPos** mode. The position is available as scaled as "diPosition" and unscaled as "rUnScaledVal".

In the **'SETTO_REFERENCE'** mode, the drive is referenced and the initial position is applied to "RefPoint".

Declaration of the example

```
PROGRAM PrgIL_IncIn
VAR
    fbIL_IncIn: IL_IncIn;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    diPosition: DINT;
    rUnScaledVal: REAL;
    bStatus_I1: BOOL;
    bStatus_I2: BOOL;
    bStatus_I3: BOOL;
    bDeviceHomed: BOOL;
END_VAR
```

Implementation of the example

```
// Configuration
  fbIL_IncIn.IncConfig.EncoderType := TRUE;
  fbIL_IncIn.IncConfig.IncPerTurn  := INT#100;
  fbIL_IncIn.IncConfig.Offset      := DINT#45;
  fbIL_IncIn.IncConfig.PathPerTurn := DINT#50;
  fbIL_IncIn.IncConfig.PulseFactor := INT#1;
  fbIL_IncIn.OpMode                := IL_INLN_INC_MODE.READPOS;
  fbIL_IncIn.RefPoint              := DINT#20;
  fbIL_IncIn.SetOutput             := FALSE;
  wOutProcessData_0             := fbIL_IncIn.OutProcData_0;
  wOutProcessData_1             := fbIL_IncIn.OutProcData_1;
  fbIL_IncIn.InProcData_0          := wInProcessData_0;
  fbIL_IncIn.InProcData_1          := wInProcessData_1;

 // Activating the function block
   fbIL_IncIn(Enable :=TRUE);

 // Read position
  IF fbIL_IncIn.InOperation = TRUE THEN
   fbIL_IncIn.OpMode  := IL_INLN_INC_MODE.READPOS;
      diPosition         := fbIL_IncIn.Position;
      rUnScaledVal       := fbIL_IncIn.IncData.IncValue;
      bStatus_I1         := fbIL_IncIn.IncData.Status_I1;
      bStatus_I2         := fbIL_IncIn.IncData.Status_I2;
      bStatus_I3         := fbIL_IncIn.IncData.Status_I3;
      bDeviceHomed       := fbIL_IncIn.IncData.DeviceHomed;
  END_IF
  // Device homing
  IF fbIL_IncIn.InOperation = TRUE THEN
      fbIL_IncIn.RefPoint := DINT#30;
      fbIL_IncIn.OpMode   := IL_INLN_INC_MODE.SETTO_REFERENCE;
```

```
    bStatus_I1          := fbIL_IncIn.IncData.Status_I1;
    bStatus_I2          := fbIL_IncIn.IncData.Status_I2;
    bStatus_I3          := fbIL_IncIn.IncData.Status_I3;
    bDeviceHomed        := fbIL_IncIn.IncData.DeviceHomed;
END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "wInProcessData_x" and "wOutProcessData_x" are declared in the program "prgIL_IncIn".

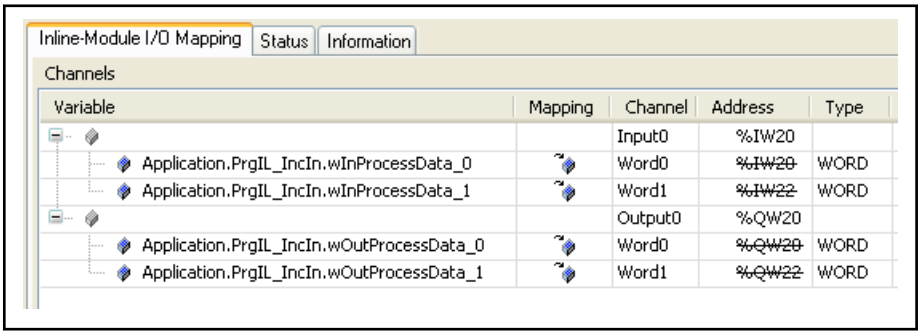


Fig. 14-57: Process data interface

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000001	“PulseFactor” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000002	“IncPerTurn” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000003	“PathPerTurn” is invalid (<=0) or (> PulseFactor * IncPerTurn)
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000004	Homing was triggered multiple times
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000005	“RefPoint” is smaller than the measuring range
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000006	“RefPoint” is higher than the measuring range
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000007	“OpMode” is invalid
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#00000008	Actual position is invalid (voltage supply)
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000A	Actual position is invalid (encoder error)
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000C	Homing not possible, parameterization error
INPUT_INVALID_ERROR (16#0001)	16#00003002	16#0000000E	Actual position invalid (boundary conditions changed during the calculation)

Tab. 14-115: Error codes of the IL_IncIn

14.5 Communication modules

14.5.1 IL_RsCom

Brief description

The **IL_RSCom** function block parameterizes the communication modules **R-IB IL RS232 PRO**, **R-IB IL RS485/422 PRO** and **R-IB IL RS-Uni** (Inline terminals for a serial data transfer) to send and receive data.

Compatibility

The function block can be used for Inline bus, Profibus, Sercos III and Profinet I/O field bus.

Interface description

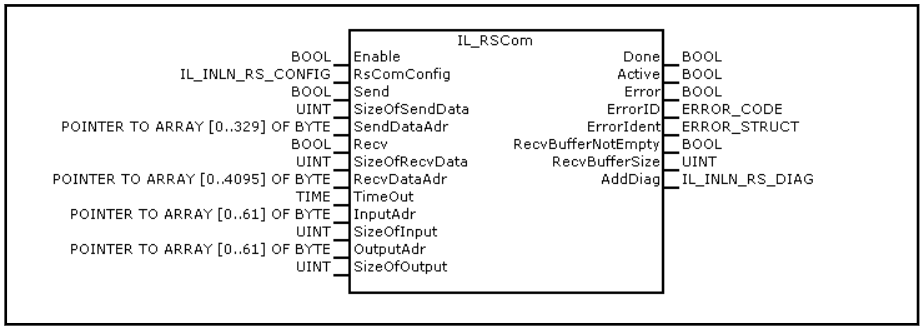


Fig. 14-58: IL_RsCom function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Continuously processes the function block when "Enable" is TRUE
	RsComConfig	IL_INLN_RS_CONFIG	Structure with variables to parameterize the module. IL_INLN_RS_CONFIG, page 689
	Send	BOOL	Data is sent at a positive edge. Transmission period is monitored using "TimeOut"
	SendDataAdr	POINTER TO ARRAY [0..329] OF BYTE	Pointer to transmission data array (SendDataAdr <= transmission data array)
	SizeOfSendData	UINT	Number of bytes to be sent.
	Recv	BOOL	Data is read at a negative edge. Reading period is monitored via "TimeOut"
	SizeOfRecvData	UINT	Number of bytes to be received.
	RecvDataAdr	POINTER TO ARRAY [0..4095] OF BYTE	Pointer to reception data array (RecvDataAdr <= reception data array)
	TimeOut	TIME	Time to monitor transmission/reception
	InputAdr	POINTER TO ARRAY [0..61] OF BYTE	Address of process input data array
	SizeOfInput	UINT	Size of process input data array
	OutputAdr	POINTER TO ARRAY [0..61] OF BYTE	Address of process output data array
	SizeOfOutput	UINT	Size of process output data array

	Name	Type	Description
VAR OUTPUT	Done	BOOL	Data transmission completed if TRUE
	Active	BOOL	Data transmission running if TRUE
	Error	BOOL	Indicates an error while the function block is processed
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Detailed error information Troubleshooting, page 683
	RecvBufferNotEmpty	BOOL	TRUE: Data available for reading in buffer
	RecvBufferSize	UINT	Number of bytes available for reading in the buffer
	AddDiag	IL_INLN_RS_DIAG	Structure for an extended diagnostics IL_INLN_RS_DIAG, page 692

Tab. 14-116: IL_RsCom interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
Send	BOOL	FALSE	TRUE	FALSE	Continuously with "Enable"
SizeOfSendData	UINT	0	330	0	Continuously with "Enable"
Recv	BOOL	FALSE	TRUE	FALSE	Continuously with "Enable"
SizeOfRecvData	UINT	0	4096	0	Continuously with "Enable"
TimeOut	TIME			T#2s	Continuously with "Enable"
SizeOfInput	UINT	12	62	0	Positive edge at "Enable"
SizeOfOutput	UINT	12	62	0	Positive edge at "Enable"

Tab. 14-117: Input behavior of IL_RsCom

Special features for handling

The process image width of the terminal R-IB IL RSUNI is determined by the DIP switch position and always contains 7, 15 or 31 data words in the direction of the input and output. The DIP switches have to be set correctly before selecting the module type.

- The RecvBufferSize is always zero in the “end-to-end” protocol. RecvBufferNotEmpty indicates that data can be read now.
- RecvBufferNotEmpty is set to TRUE in the transparent protocol when there is data in the input buffer. The module requires some cycles until all data is received. During this period, RecvBufferSize changes multiple times.

  Recommended workaround for the user when receiving data:

  Check the RecvBufferSize output for changes. If the number of received bytes does not change anymore, data can be received.

The RecvBufferSize output can also be compared with the amount of data expected (number of bytes) until the amount of data expected (number of bytes) is reached and then receive data.

If data is received without the RecvBufferSize output being checked, data is received in multiple split sequences in case of multiple function block calls.

Functional Description

The function block parameterizes the communication modules **R-IB IL RS232 PRO**, **R-IB IL RS485/422 PRO** and **R-IB IL RS-Uni** to send and receive data.

- **Transparent:** In case of the "Transparent" protocol, serial data is transmitted through the terminal as provided by the serial interface or the bus.
- **XON/XOFF:** This protocol operates similarly to the "Transparent" protocol. However, the transmission is controlled via XON/XOFF characters.
- **END-END:** Serial data is prepared for the end-to-end protocol.

  If serial data is sent from the bus, two additional characters, the first and the second delimiter, are attached for the transmission to the serial interface. The first and second delimiter have to be defined while configuring the terminal.

  Received data can only be read if both delimiters were received
- **Dual buffer protocol:** The latest data block received is saved.

  A data block is defined as a sequence of characters with the terminators first and second delimiter as in case of the end-to-end protocol. When a new data block is received, the previous one is overwritten. This is caused by two buffers on which it is alternatingly written on. Thus, one buffer is always ready to receive the serial data while the second one stores the latest received data block. Only after both delimiters were detected subsequently, the data block is complete and can be read by the bus (not available with **R-IB IL RS-Uni**).
- **3964R:** This protocol is the most time-consuming one and was developed by Siemens. It uses a start and end ID, a checksum and time monitoring. Three buffers each are available for transmission and reception (not available with **R-IB IL RS-Uni**).

The following table provides an overview on the sizes of the transmission and reception buffers for the modules RS232, RS485 and RSUni.

Protocol	RS232 reception buffer size	RS232 sending buffer size	RS485 reception buffer size	RS485 sending buffer size
TRANSPARENT	4096 bytes	330 bytes	4096 bytes	330 bytes
END-END:	3 buffers each with 330 bytes	330 bytes	25 buffers each with 11 bytes	330 bytes
DUAL BUFFER	2 buffers each with 330 bytes	330 bytes	2 buffers each with 11 bytes	330 bytes
3964R	3 buffers each with 330 bytes	255 bytes	25 buffers each with 11 bytes	255 bytes
XON/XOFF	4096 bytes	330 bytes	4096 bytes	330 bytes

☞ In case of the **R-IB IL RS-Uni** module, note that the process data depends on the position of the DIP switch. Check whether the DIP switch position is 7 Word, 15 Word or 31 Word.

Example 1 The function block is used to parameterize RS232 PRO , RS485 PRO and RSUni modules for different protocols such as "Transparent", "End-to-end", "Dual buffer", 3964R and XON-XOFF.

An RS232 module with "Transparent" protocol is configured in the following. The transparent protocol has 4096 bytes of reception buffer and 330 bytes of transmission buffer. It is mandatory to specify the addresses of transmission and reception arrays as well as of process data before calling the function block. Otherwise, the NULL pointer access results and the function block generates an error. The number of data received has to be equal to or smaller than the reception data array.

When "Enable" = TRUE, the function block configures the module and starts processing: The function block reads the device state If no error results, data handling is started.

Declaration of the example

```
PROGRAM PrgIL_RsCom
VAR
    fbIL_RSCom : IL_RSCom;
    arSendDataArray: ARRAY[1..330] OF BYTE;
    arRecvDataArray: ARRAY[1..4096] OF BYTE;
    arInProcessData: ARRAY[0..11] OF BYTE;
    arOutProcessData: ARRAY[0..11] OF BYTE;
    bRecvBufferNotEmpty: BOOL;
    uiRecvBufferSize: UINT;
    bRecvBufferFull: BOOL;
    bSendBufferFull: BOOL;
    bSendBufferNotEmpty: BOOL;
    wFirmware: WORD;
 uiExpectedDataSize: UINT := UINT#330;
    bRecvError: BOOL;
    bSendError: BOOL;
 bSend: BOOL;
    bRecv: BOOL;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Implementation of the example

```
// Configuration
 fbIL_RSCom.RsComConfig.Baudrate := IL_INLN_RS_BAUDRATE.BAUDRATE_9600;
 fbIL_RSCom.RsComConfig.DataWidth := IL_INLN_RS_DATAWIDTH.DATABIT7_EVEN_STOPBIT1;
 fbIL_RSCom.RsComConfig.Protocol := IL_INLN_RS_PROTOCOL.TRANSPARENT ;
 fbIL_RSCom.RsComConfig.ErrorPattern := 16#24;
 fbIL_RSCom.RsComConfig.FirstDelimiter := 16#0D ;
 fbIL_RSCom.RsComConfig.SecondDelimiter := 16#0A ;
 fbIL_RSCom.RsComConfig.InterfaceType := IL_INLN_RS_INTERFACE.RS232 ;
 fbIL_RSCom.RsComConfig.ModuleType := IL_INLN_RS_MODULETYPE.RS232PRO ;

 // FB invocation
   fbIL_RSCom(
   Enable:=TRUE,
   Send:= bSend,
   SizeOfSendData:= SIZEOF(arSendDataArray),
   SendDataAdr:= ADR(arSendDataArray),
   Recv:= bRecv,
   SizeOfRecvData:= SIZEOF(arRecvDataArray),
   RecvDataAdr:= ADR(arRecvDataArray),
   TimeOut:= T#2S,
   InputAdr:= ADR(arInProcessData),
   SizeOfInput:= SIZEOF(arInProcessData),
   OutputAdr:= ADR(arOutProcessData),
   SizeOfOutput:= SIZEOF(arOutProcessData),
   Done=> bDone,
   Active=> bActive,
   Error=> bError,
   ErrorID=> enErrorID,
   ErrorIdent=> stErrorIdent,
   RecvBufferNotEmpty=> bRecvBufferNotEmpty,
   RecvBufferSize=> uiRecvBufferSize);
```

```
 // Status monitoring
 IF    fbIL_RSCom.Active = TRUE THEN
   bRecvBufferFull := fbIL_RSCom.AddDiag.RecvBufferFull ;
   bSendBufferFull := fbIL_RSCom.AddDiag.SendBufferFull ;
   bSendBufferNotEmpty := fbIL_RSCom.AddDiag.SendBufferNotEmpty ;
   wFirmware := fbIL_RSCom.AddDiag.Firmware ;
   bRecvError := fbIL_RSCom.AddDiag.ErrorRecv ;
   bSendError := fbIL_RSCom.AddDiag.ErrorSend ;
 END_IF

// Receive cycle
 IF    (fbIL_RSCom.Active = TRUE) AND (fbIL_RSCom.RecvBufferNotEmpty = TRUE)
    AND (fbIL_RSCom.RecvBufferSize >= uiExpectedDataSize) THEN
 bRecv := TRUE ;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "arInProcessData" and "arOutProcessData" are declared in the "PrgIL_RsCom" program.

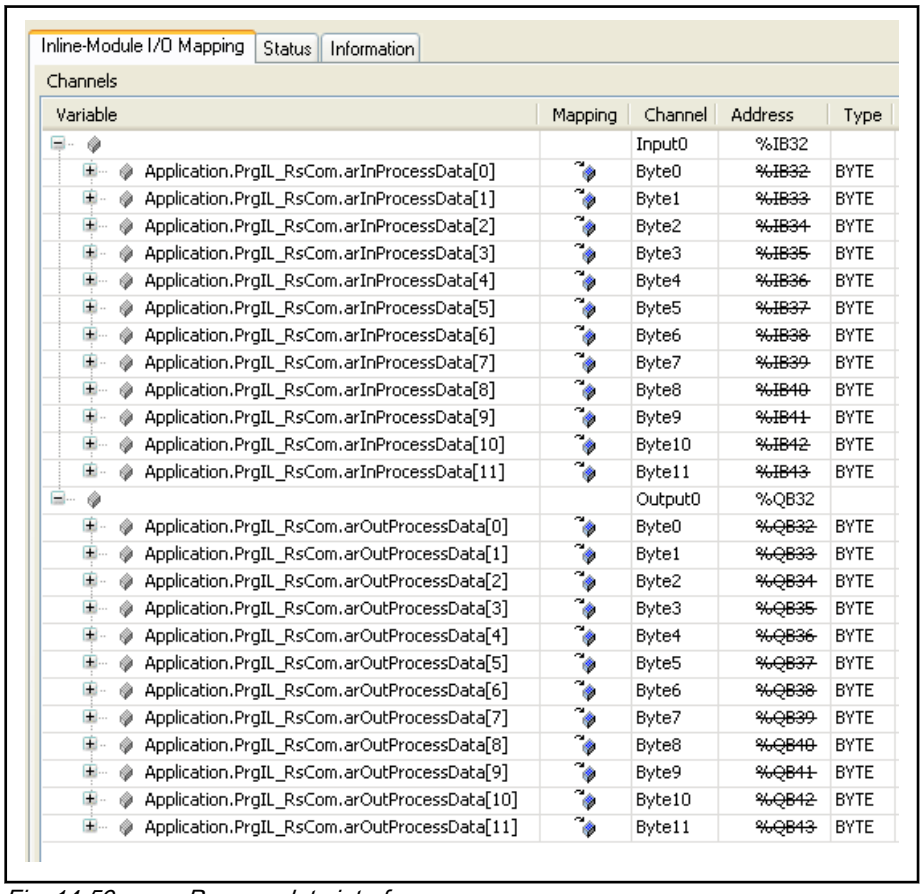


Fig. 14-59: Process data interface

Example 2

An RSUni module with "Transparent" protocol is configured in the following. The "Transparent" protocol has 4096 bytes of reception buffer and 330 bytes of transmission buffer. It is mandatory to specify the addresses of transmission and reception arrays as well as of process data before calling the function block. Otherwise, the NULL pointer access results and the function block generates an error. The number of data received has to be equal to or smaller than the reception data array.

When "Enable" = TRUE, the function block configures the module and starts processing: The function block reads the device state If no error results, data handling is started.

Declaration of the example

```
PROGRAM PrgIL_RsCom
VAR
    fbIL_RSComUni : IL_RSCom;
    bEnableUni: BOOL;
    arSendDataArrayUni: ARRAY[1..330] OF BYTE;
    arRecvDataArrayUni: ARRAY[1..4096] OF BYTE;
    arInProcessDataUni: ARRAY[0..13] OF BYTE;
    arOutProcessDataUni: ARRAY[0..13] OF BYTE;
    bRecvBufferNotEmptyUni: BOOL;
    uiRecvBufferSizeUni: UINT;
 uiExpectedDataSize: UINT := UINT#330;
    bRecvBufferFullUni: BOOL;
    bSendBufferFullUni: BOOL;
    bSendBufferNotEmptyUni: BOOL;
    bSendUni: BOOL;
    bRecvUni: BOOL;
    bDoneUni: BOOL;
    bActiveUni: BOOL;
    bErrorUni: BOOL;
    enErrorIDUni: ERROR_CODE;
    stErrorIdentUni: ERROR_STRUCT;
END_VAR
```

Implementation of the example

```
//Configuration
fbIL_RSComUni.RsComConfig.Baudrate := IL_INLN_RS_BAUDRATE.BAUDRATE_9600;
fbIL_RSComUni.RsComConfig.DataWidth := IL_INLN_RS_DATAWIDTH.DATABIT8_EVEN_
                                       STOPBIT1;
fbIL_RSComUni.RsComConfig.Protocol := IL_INLN_RS_PROTOCOL.TRANSPARENT ;
fbIL_RSComUni.RsComConfig.ErrorPattern := 16#24;
fbIL_RSComUni.RsComConfig.FirstDelimiter := 16#0D ;
fbIL_RSComUni.RsComConfig.SecondDelimiter := 16#0A ;
fbIL_RSComUni.RsComConfig.InterfaceType := IL_INLN_RS_INTERFACE.RS485 ;
fbIL_RSComUni.RsComConfig.ModuleType := IL_INLN_RS_MODULETYPE.RS485PRO ;

//Invocation
fbIL_RSComUni(
    Enable:= bEnableUni,
    Send:= bSendUni,
    SizeOfSendData:= SIZEOF(arSendDataArrayUni),
    SendDataAdr:= ADR(arSendDataArrayUni),
    Recv:= bRecvUni,
    SizeOfRecvData:= SIZEOF(arRecvDataArrayUni),
    RecvDataAdr:= ADR(arRecvDataArrayUni),
    TimeOut:= T#2S,
    InputAdr:= ADR(arInProcessDataUni),
    SizeOfInput:= SIZEOF(arInProcessDataUni),
    OutputAdr:= ADR(arOutProcessDataUni),
    SizeOfOutput:= SIZEOF(arOutProcessDataUni),
    Done=> bDoneUni,
    Active=> bActiveUni,
    Error=> bErrorUni,
    ErrorID=> enErrorIDUni,
    ErrorIdent=> stErrorIdentUni,
    RecvBufferNotEmpty=> bRecvBufferNotEmptyUni,
    RecvBufferSize=> uiRecvBufferSizeUni );

// Receive cycle
 IF    (fbIL_RSCom.Active = TRUE) AND (fbIL_RSCom.RecvBufferNotEmpty = TRUE)
   AND (fbIL_RSCom.RecvBufferSize >= uiExpectedDataSize) THEN
   bRecv := TRUE ;
 END_IF
```

Process data interface of the example

The process input and output words of the function block have to be mapped to the process data words of the individual field bus system.

The variables "arInProcessData" and "arOutProcessData" are declared in the "PrgIL_RsCom" program.

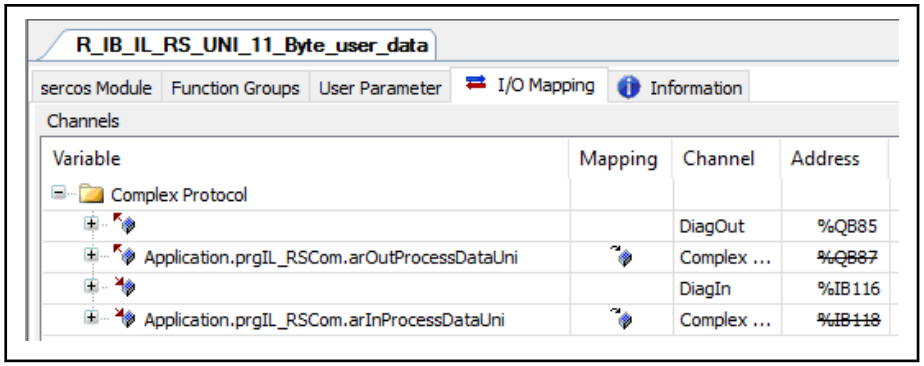


Fig. 14-60: Process data interface

Time behavior

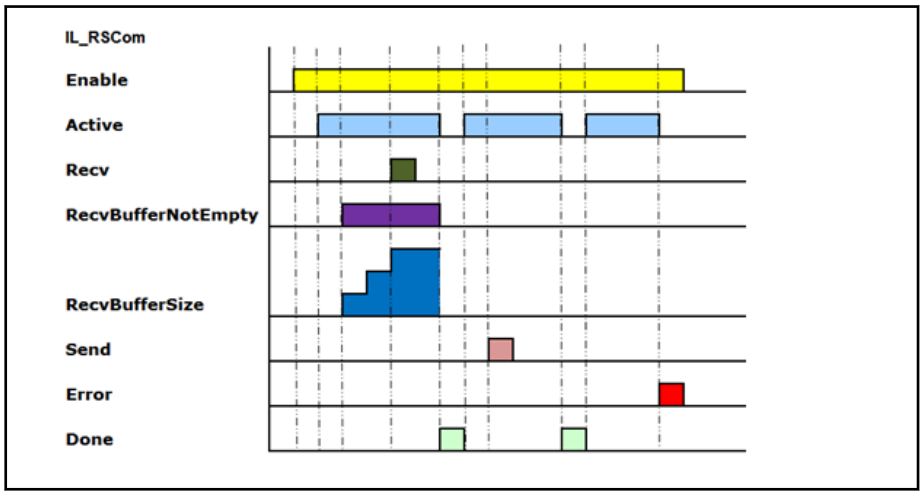


Fig. 14-61: Time behavior of IL_RSCom

Troubleshooting

The function block uses the error table INLINEIO_TABLE, 16#0190. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#3002	16#0001	Baud rate is invalid
INPUT_INVALID_ERROR (16#0001)	16#3002	16#0002	Data width is invalid
INPUT_INVALID_ERROR (16#0001)	16#3002	16#0003	Protocol is invalid
INPUT_INVALID_ERROR (16#0001)	16#3002	16#0004	Interface input is invalid
INPUT_INVALID_ERROR (16#0001)	16#3002	16#0005	SizeOfSendData is invalid
INPUT_INVALID_ERROR (16#0001)	16#3002	16#0006	SizeOfRecvData is invalid
INPUT_INVALID_ERROR (16#0001)	16#3010	16#0000	Wrong module connected
COMMUNICATION_ERROR (16#0002)	16#3011	16#0000	Transmission period elapsed
COMMUNICATION_ERROR (16#0002)	16#3011	16#0001	Error while transmitting data

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR (16#0002)	16#3011	16#0002	Error while transferring transmission data to buffer
COMMUNICATION_ERROR (16#0002)	16#3011	16#0003	Transmission error in the 3964R protocol
COMMUNICATION_ERROR (16#0002)	16#3012	16#0000	Reception period elapsed
COMMUNICATION_ERROR (16#0002)	16#3012	16#0001	Error while receiving data
COMMUNICATION_ERROR (16#0002)	16#3012	16#0002	Reception error in the 3964R protocol
RESOURCE_ERROR (16#0003)	16#3013	16#0002	NULL pointer, invalid input address
DEVICE_ERROR (16#0008)	16#3020	16#0000	Time elapsed, device does not response
DEVICE_ERROR (16#0008)	16#3020	16#0001	Invalid parameters for the specified command or failure of the peripheral voltage

Tab. 14-118: Error codes of the IL_RsCom

14.6 Data types

14.6.1 IL_INLN_AI_CONFIG

The structure supports the configuration of a channel of an analog input module.

Type definition

```
TYPE IL_INLN_AI_CONFIG :
STRUCT
 HighLimit : REAL := 10.0 ; // High limit of the normalized value
 LowLimit : REAL :=0.0; // Low limit of the normalized value
 NoOfSamples : BYTE := BYTE#16 ; // Average value 0, 4, 16, 32
 InputRange : IL_INLN_AI_RANGE ; // Enum to set the Measuring range
END_STRUCT
END_TYPE
```

Structure and enumeration types used

- IL_INLN_AI_RANGE

Used in

- IL_AI2Sf
- IL_AI4Ef
- IL_AI4IU
- IL_AI8SfIs

14.6.2 IL_INLN_AI_RANGE

Use this enumeration to specify the measuring range of individual input channels.

Type definition

```
TYPE IL_INLN_AI_RANGE :
(
   UNIPOLAR10V     :=0,  // Measuring Voltage 0 to 10 V
   BIPOLAR10V      :=1,  // Measuring Voltage -10 to +10 V
   UNIPOLAR5V      :=2,  // Measuring Voltage 0 to 5 V
   BIPOLAR5V       :=3,  // Measuring Voltage -5 to +5 V
   UNIPOLAR25V     :=4,  // Measuring Voltage 0 to 25 V
   BIPOLAR25V      :=5,  // Measuring Voltage -25 to + 25 V
```

```
   UNIPOLAR50V     :=6,  // Measuring Voltage 0 to 50 V
   UNIPOLAR20MA    :=8,  // Measuring Current 0 to 20 mA
   BIPOLAR20MA     :=9,  // Measuring Current -20 to +20 mA
   UNIPOLAR4TO20MA :=10, // Measuring Current 4 to 20 mA
   UNIPOLAR40MA    :=12, // Measuring Current 0 to 40 mA
   BIPOLAR40MA     :=13  // Measuring Current -40 to +40 mA
);
END_TYPE
```

Used in

- IL_AI2Sf
- IL_AI4Ef
- IL_AI4IU
- IL_AI8SfIs

14.6.3 IL_INLN_AI_DATA

IL_INLN_AI_DATA The structure contains the output parameters of one channel.

The parameter can be determined for each channel as follows:

`ParameterName[Channel_Number].[Function]`

e.g. `OutputData[0].NormalizedValue`.

Name	Type	Description
NormalizedValue	REAL	Normalized value between upper and lower limit value
DiagCode	IL_INLN_DIAG_CODE	Diagnostic code

Tab. 14-119: Structural elements IL_INLN_AI_DATA

14.6.4 IL_INLN_DIAG_CODE

The table contains channel-related errors and diagnostics output by the function blocks of the library.

DiagCode	ENUM	Description
16#8001	OVERRANGE	Overrange
16#8002	OPEN_CIRCUIT	Open circuit, the output is configured as current output and the set current cannot flow
16#8003	OVER_LOAD	Short-circuit/overload, the output is configured as voltage output and is overloaded
16#8004	MEASURED_VALUE_INVALID	Measured value invalid / No valid measured value
16#8008	COLD_JUNCTION_DEFECTIVE	Cold junction defective
16#8010	INVALID_CONFIGURATION	Invalid configuration
16#8020	ERROR_IN_IO_SUPPLY_VOLTAGE	I/O supply voltage faulty
16#8040	MODULE_FAULTY	Module errors
16#8080	UNDERRANGE	Underrange

Tab. 14-120: Enum IL_INLN_DIAG_CODE

Used in

- IL_AI2Sf
- IL_AI4Ef
- IL_AI4IU

- IL_AI8SfIs
- IL_AO1Sf
- IL_AO2SfBp
- IL_Sgi2f
- IL_Temp2Rtd
- IL_Temp2Uth
- IL_Temp4_8Rtd

14.6.5 IL_INLN_CNT_INCONFIG

An input filter or input logic to link two input signals can be configured using the enumeration.

Type definition

```
TYPE IL_INLN_CNT_INCONFIG :
(
 NO_INPUT_CONFIG := 0, //No Input Configuration
 FILTER_FOR_MECH_SWITCH := 1, //Source and Gate filter for Mechanical Contacts
 PULSE_GENERATOR := 2, //Setting for pulse generator
 SOURCE_GATE_OPERATION := 3 //Souce Gate Logic Operation as per "IL_INLN_CNT_SGLOGIC"
);
END_TYPE
```

Used in

- IL_CntCount
- IL_CntFrequency
- IL_CntTime

14.6.6 IL_INLN_TEMP_CONFIG

The structure supports the configuration of module channels to detect temperatures.

Type definition

```
TYPE IL_INLN_TEMP_CONFIG :
STRUCT
 NoOfSamples : BYTE := BYTE#16 ; // Average value 0, 4, 16, 32
 SensorType : IL_INLN_TEMP_SENSOR; // Sensor Type Selection
 ConnectionType : IL_INLN_TEMP_CONNECTIONTYPE; // Sensor Wire Connection Selection
 Resolution : WORD := 0; // Resolution of value display
 Resistance : IL_INLN_TEMP_R0 := IL_INLN_TEMP_R0.R100; // Value of reference Resistance.
END_STRUCT
END_TYPE
```

Structure and enumeration types used

- IL_INLN_TEMP_SENSOR, ENUM, sensor types
- IL_INLN_TEMP_CONNECTIONTYPE, ENUM, connection types of sensors
- IL_INLN_TEMP_R0, ENUM, selection of resistance reference values

Used in

- IL_INLN_TEMP_CONFIG, IL_Temp2Rtd
- IL_INLN_TEMP_CONFIG, IL_Temp4_8Rtd

14.6.7 IL_INLN_TEMP_CONNECTIONTYPE

The sensor connection type to configure a channel can be selected using this enumeration type.

Type definition

```
TYPE IL_INLN_TEMP_CONNECTIONTYPE :
(
 THREEWIRE := 0, // Connection type sensor is 3 wire type
 TWOWIRE := 16#1000, // Connection type sensor is 2 wire type
 FOURWIRE := 16#2000 // Connection type sensor is 4 wire type
 );
END_TYPE
```

Used in

- IL_INLN_TEMP_SENSOR, IL_Temp2Rtd
- IL_INLN_TEMP_SENSOR, IL_Temp4_8Rtd

14.6.8 IL_INLN_TEMP_R0

The resistance reference value to configure the channel can be selected using this enumeration type.

Type definition

```
TYPE IL_INLN_TEMP_R0 :
(
 R100 := 0, // Resistance of Sensor at 0deg : 100 ohm
 R10 := 1, // Resistance of Sensor at 0deg : 10 ohm
 R20 := 2, // Resistance of Sensor at 0deg : 20 ohm
 R30 := 3, // Resistance of Sensor at 0deg : 30 ohm
 R50 := 4, // Resistance of Sensor at 0deg : 50 ohm
 R120 := 5, // Resistance of Sensor at 0deg : 120 ohm
 R150 := 6, // Resistance of Sensor at 0deg : 150 ohm
 R200 := 7, // Resistance of Sensor at 0deg : 200 ohm
 R240 := 8, // Resistance of Sensor at 0deg : 240 ohm
 R300 := 9, // Resistance of Sensor at 0deg : 300 ohm
 R400 := 10, // Resistance of Sensor at 0deg : 400 ohm
 R500 := 11, // Resistance of Sensor at 0deg : 500 ohm
 R1000 := 12, // Resistance of Sensor at 0deg : 1000 ohm
 R1500 := 13, // Resistance of Sensor at 0deg : 1500 ohm
 R2000 := 14, // Resistance of Sensor at 0deg : 2000 ohm
 R10000 := 15, // Resistance of Sensor at 0deg : 10000 ohm
 R3000 := 16 // Resistance of Sensor at 0deg : 3000 ohm
)WORD;
END_TYPE
```

Used in

- IL_INLN_TEMP_SENSOR, IL_Temp2Rtd
- IL_INLN_TEMP_SENSOR, IL_Temp4_8Rtd

14.6.9 IL_INLN_TEMP_SENSOR

The sensor type to configure a channel can be selected using this enumeration type.

Type definition

```
TYPE IL_INLN_TEMP_SENSOR :
(
    PTDIN       := 0,    //  PT DIN SENSOR TYPE
    PTSAMA      := 1,    //  PT SAMA SENSOR TYPE
    NIDIN       := 2,    //  NI DIN SENSOR TYPE
    NISAMA      := 3,    //  NI SAMA SENSOR TYPE
    CU10        := 4,    //  CU 10 SENSOR TYPE
    CU50        := 5,    //  CU 50 SENSOR TYPE
    CU53        := 6,    //  CU 53 SENSOR TYPE
    NI1000      := 7,    //  NI 1000 (LANDIS & GYR) SENSOR TYPE
    NI500       := 8,    //  NI 500 (VIESSMANN) SENSOR TYPE
    KTY81110    := 9,    //  KTY81 -110 SENSOR TYPE
    KTY84       := 10,   //  KTY84 SENSOR TYPE
    KTY81210    := 11,   //  KTY81-210 SENSOR TYPE
    POT         := 13,   //  POTENTIOMETER(%) SENSOR TYPE
    LINEAR400   := 14,   //  LINEAR R : 0 TO 400 Ohm SENSOR TYPE
    LINEAR20000 := 15,   //  LINEAR R :0 TO 20000 Ohm SENSOR TYPE
    LINEAR4000  := 16    //  LINEAR R :0 TO 4000 Ohm SENSOR TYPE
 );
END_TYPE
```

Used in

- IL_INLN_TEMP_SENSOR, IL_Temp2Rtd
- IL_INLN_TEMP_SENSOR, IL_Temp4_8Rtd

14.6.10 IL_INLN_TEMP_EF_CONFIG

The structure supports the configuration of module channels to detect temperatures.

Type definition

```
TYPE IL_INLN_TEMP_EF_CONFIG :
STRUCT
 SensorType     : IL_INLN_TEMP_EF_SENSOR:= IL_INLN_TEMP_EF_SENSOR.PTDIN; // Sensor Type Selection
 Resolution     : BYTE:= BYTE#0;                                         // Resolution of value display
 Resistance     : IL_INLN_TEMP_EF_R0 := IL_INLN_TEMP_EF_R0.R100; // Value of reference Resistance
 FilterTime     : BYTE:= BYTE#0;                                         // Filter Time
 ODSSensitivity : IL_INLN_TEMP_EF_ODS;                                    //Open circuit detection sensitivity
END_STRUCT
END_TYPE
```

Structure and enumeration types used

- IL_INLN_TEMP_EF_SENSOR, ENUM, sensor types
- IL_INLN_TEMP_EF_R0, ENUM, selection of resistance reference values
- IL_INLN_TEMP_EF_ODS, ENUM, sensitivity

Used in

- IL_INLN_TEMP_EF_CONFIG, IL_Temp4_8RtdEf

14.6.11 IL_INLN_TEMP_EF_ODS

This enumeration type allows to select the sensitivity of the wire break detection. (ODS - **o**pen circuit **d**etection **s**ensitivity) : "High sensitivity"/ "Average sensitivity"/ "Off".

Type definition

```
TYPE IL_INLN_TEMP_EF_ODS:
(
HIGH_SENSITIVITY := 0   // High Sensitivity
MEDIUM_SENSITIVITY := 1 // Medium Sensitivity
SWITCHED_OFF := 3       // Switched Off
) BYTE;
END_TYPE
```

Used in

- IL_INLN_TEMP_EF_CONFIG, IL_Temp4_8RtdEf

14.6.12 IL_INLN_TEMP_EF_R0

The resistance reference value to configure the channel can be selected using this enumeration type.

Type definition

```
TYPE IL_INLN_TEMP_EF_R0 :
(
    R100   := 0,    // Resistance of Sensor at 0deg  : 100 ohm
    R10    := 1,    // Resistance of Sensor at 0deg  : 10 ohm
    R20    := 2,    // Resistance of Sensor at 0deg  : 20 ohm
    R30    := 3,    // Resistance of Sensor at 0deg  : 30 ohm
    R50    := 4,    // Resistance of Sensor at 0deg  : 50 ohm
    R120   := 5,    // Resistance of Sensor at 0deg  : 120 ohm
    R150   := 6,    // Resistance of Sensor at 0deg  : 150 ohm
    R200   := 7,    // Resistance of Sensor at 0deg  : 200 ohm
    R240   := 8,    // Resistance of Sensor at 0deg  : 240 ohm
    R300   := 9,    // Resistance of Sensor at 0deg  : 300 ohm
    R400   := 10,   // Resistance of Sensor at 0deg  : 400 ohm
    R500   := 11,   // Resistance of Sensor at 0deg  : 500 ohm
```

```
    R1000  := 12,   // Resistance of Sensor at 0deg  : 1000 ohm
    R1500  := 13,   // Resistance of Sensor at 0deg  : 1500 ohm
    R2000  := 14,   // Resistance of Sensor at 0deg  : 2000 ohm
    R10000 := 15,   // Resistance of Sensor at 0deg  : 10000 ohm
    R3000  := 16    // Resistance of Sensor at 0deg  : 3000 ohm
) WORD;
END_TYPE
```

Used in

- IL_INLN_TEMP_EF_CONFIG, IL_Temp4_8RtdEf

14.6.13 IL_INLN_TEMP_EF_SENSOR

The sensor type to configure a channel can be selected using this enumeration type.

Type definition

```
TYPE IL_INLN_TEMP_EF_SENSOR :
(
    PTDIN       := 0,    //  Pt DIN SENSOR TYPE
    PTSAMA      := 1,    //  Pt SAMA SENSOR TYPE
    NIDIN       := 2,    //  Ni DIN SENSOR TYPE
    NISAMA      := 3,    //  Ni SAMA SENSOR TYPE
    CU10        := 4,    //  Cu 10 SENSOR TYPE
    CU50        := 5,    //  Cu 50 SENSOR TYPE
    CU53        := 6,    //  Cu 53 SENSOR TYPE
    NI1000      := 7,    //  Ni 1000 (LANDIS & GYR) SENSOR TYPE
    NI500       := 8,    //  Ni 500 (VIESSMANN) SENSOR TYPE
    KTY81110    := 9,    //  KTY81 -110 SENSOR TYPE
    KTY84       := 16#A, //  KTY84 SENSOR TYPE
    KTY81210    := 16#B, //  KTY81-210 SENSOR TYPE
    LINEAR30000 := 16#C, //  LINEAR R : 0 TO 30000 Ohm SENSOR TYPE

    LINEAR500   := 16#E, //  LINEAR R :0 TO 500 Ohm SENSOR TYPE
    LINEAR5000  := 16#F, //  LINEAR R :0 TO 5000 Ohm SENSOR TYPE
 ) WORD;
END_TYPE
```

Used in

- IL_INLN_TEMP_EF_CONFIG, IL_Temp4_8RtdEf

14.6.14 IL_INLN_RS_CONFIG

The structure describes the configuration parameters of the following modules: **R-IB IL RS232 PRO**, **R-IB IL RS485/422 PRO** and **R-IB IL RS-Uni**. It also includes the default data of a COM interface.

Name	Type	Description	Min. value	Max. value	Default value	Effective
Protocol	IL_INLN_RS_PROTOCOL	Protocol type IL_INLN_RS_PROTOCOL, page 690	TRANSPARENT	XON_XOFF	TRANSPARENT	Positive edge at "Enable"
Baud rate	IL_INLN_RS_BAUDRATE	Baud rate IL_INLN_RS_BAUDRATE, page 690	BAUDRATE_110	BAUDRATE_250000	BAUDRATE_9600	Positive edge at "Enable"
DataWidth	IL_INLN_RS_DATAWIDTH	Data width IL_INLN_RS_DATAWIDTH, page 691	DATABIT7_EVEN_STOPBIT1	DATABIT6_NONE_STOPBIT1	DATABIT7_EVEN_STOPBIT1	Positive edge at "Enable"
ErrorPattern	BYTE	Any character			16#24	Positive edge at "Enable"

Name	Type	Description	Min. value	Max. value	Default value	Effective
FirstDelimiter	BYTE	Any character (used for end-to-end and dual buffer protocol)			16#0D	Positive edge at "Enable"
SecondDelimiter	BYTE	Any character (used for end-to-end and dual buffer protocol)			16#0A	Positive edge at "Enable"
InterfaceType	IL_INLN_RS_INTERFACE	Interface type, (RS232, RS485 or RS422) IL_INLN_RS_INTERFACE, page 691	RS232	RS422	RS232	Positive edge at "Enable"
ModuleType	IL_INLN_RS_MODULE-TYPE	Module type (RS232, RS485 or RSUni) IL_INLN_RS_MODULETYPE, page 692	RS232PRO	RSUNI_31WORDS	RS232PRO	Positive edge at "Enable"

Tab. 14-121: IL_INLN_RS_CONFIG, IL_RsCom

14.6.15 IL_INLN_RS_PROTOCOL

The enumeration type selects the protocol type.

ENUM	Description
TRANSPARENT	Transparent protocol
END_TO_END	End-to end protocol
DUAL_BUFFER	Dual buffer protocol (not possible with **R-IB IL RS-Uni**)
LOW_PRIORITY_3964R	3964R at low priority (not possible with **R-IB IL RS-Uni**)
HIGH_PRIORITY_3964R	3964R at high priority (not possible with **R-IB IL RS-Uni**)
XON_XOFF	XON/XOFF protocol

14.6.16 IL_INLN_RS_BAUDRATE

The enumeration type supports the baud rate selection.

ENUM	Description
BAUDRATE_110	110 bauds
BAUDRATE_300	300 bauds
BAUDRATE_600	600 bauds
BAUDRATE_1200	1200 bauds
BAUDRATE_1800	1800 bauds
BAUDRATE_2400	2400 bauds
BAUDRATE_4800	4800 bauds
BAUDRATE_9600	9600 bauds

ENUM	Description
BAUDRATE_15625	15625 bauds (only **R-IB IL RS-Uni**)
BAUDRATE_19200	19200 bauds
BAUDRATE_38400	38400 bauds
BAUDRATE_57600	57600 bauds (only **R-IB IL RS-Uni**)
BAUDRATE_115200	115200 bauds (only **R-IB IL RS-Uni**)
BAUDRATE_230400	230400 bauds (only **R-IB IL RS-Uni**)
BAUDRATE_250000	250000 bauds (only **R-IB IL RS-Uni**)

14.6.17 IL_INLN_RS_DATAWIDTH

The enumeration type specifies the data width.

ENUM	Description
DATABIT7_EVEN_STOPBIT1	7 data bits, even parity, 1 stop bit
DATABIT7_ODD_STOPBIT1	7 data bits, uneven parity, 1 stop bit
DATABIT8_EVEN_STOPBIT1	8 data bits, even parity, 1 stop bit
DATABIT8_ODD_STOPBIT1	8 data bits, uneven parity, 1 stop bit
DATABIT8_NONE_STOPBIT1	8 data bits, without parity, 1 stop bit
DATABIT7_NONE_STOPBIT1	7 data bits, without parity, 1 stop bit
DATABIT7_EVEN_STOPBIT2	7 data bits, even parity, 2 stop bits
DATABIT7_ODD_STOPBIT2	7 data bits, uneven parity, 2 stop bits
DATABIT8_EVEN_STOPBIT2	8 data bits, even parity, 2 stop bits
DATABIT8_ODD_STOPBIT2	8 data bits, uneven parity, 2 stop bits
DATABIT8_NONE_STOPBIT2	8 data bits, without parity, 2 stop bits
DATABIT7_NONE_STOPBIT2	7 data bits, without parity, 2 stop bits
DATABIT8_CONSTANT0_STOP-BIT1	8 data bits, constant at 0, 1 stop bit (only **R-IB IL RS-Uni**)
DATABIT8_CONSTANT1_STOP-BIT1	8 data bits, constant at 0, 1 stop bit (only **R-IB IL RS-Uni**)
DATABIT6_NONE_STOPBIT1	6 data bits, without parity, 1 stop bit (only **R-IB IL RS-Uni**)

14.6.18 IL_INLN_RS_INTERFACE

The enumeration type supports the selection of the interface type.

ENUM	Description
RS232	RS-232 interface type
RS485	RS-485 interface type
RS422	RS-422 interface type

14.6.19 IL_INLN_RS_MODULETYPE

The enumeration types supports the selection of the module type used.

ENUM	Description
RS232PRO	Use of the R-IB IL RS232 PRO module
RS485PRO	Use of the R-IB IL RS485/422 PRO module
RSUNI_7WORDS	For the use of the R-IB IL RS-Uni module, refer to the table with DIP switch setting
RSUNI_15WORDS	For the use of the R-IB IL RS-Uni module, refer to the table with DIP switch setting
RSUNI_31WORDS	For the use of the R-IB IL RS-Uni module, refer to the table with DIP switch setting

Tab. 14-122: IL_INLN_RS_MODULETYPE

DIP switch 2	DIP switch 1	Data width	User data
OFF	OFF	7 Word	11 bytes
OFF	ON	15 Word	27 bytes
ON	OFF	31 Word	59 bytes
ON	ON	Reserved	Reserved

Tab. 14-123: DIP switch setting

14.6.20 IL_INLN_RS_DIAG

The structure contains additional diagnostic parameters such as **ErrorRecv**, **ErrorSend**, **RecvBufferfull**, **SendBufferfull**, **SendBufferNotEmpty** and the firmware version of the module.

Name	Type	Description
ErrorRecv	BOOL	Timeout while receiving a telegram
ErrorSend	BOOL	Timeout while sending a telegram
RecvBufferfull	BOOL	Receiving buffer is full
SendBufferfull	BOOL	Sending buffer is full
SendBufferNotEmpty	BOOL	Sending buffer is not empty
RecvCounter	UINT	Number of data received
Firmware	WORD	Firmware version of the module

15 RIH_Safety.library

The RIH_Safety library provides access options from the standard application to data of the SafeLogic safety control

The RIH_Safety library is automatically provided in the library manager of the standard application as soon as a SafeLogic is connected to the standard control. The global variable "SafetyData" can access SafeLogic information in the standard application. The point of entry is `SafetyData.SafetyFM`.

Moreover, the RIH_Safety library contains the following types in the "DiagTypes" folder:

- IH_S20_DIAG_ADD_VALUE contains the additional diagnostic code and the selected texts
- IH_S20_SAFETY_CHANNEL contains the channel and the input assignment
- IH_SAFETY_IO_DIAG_TEXT contains diagnostic text and diagnostic code

These types are used by the acknowledgement function blocks in theRIH_Safety_ProfibusDP, RIH_Safety_ProfinetIO and RIH_Safety_SercosIII libraries.

16 RIH_Safety_ProfibusDP.library

Introduction and overview: RIH_Safety_ProfibusDP, RIH_Safety_ProfinetIO and RIH_Safety_SercosIII

Field bus-specific functionalities of SafeLogic for use in the standard application are bundled in these libraries. The field bus assignment is identified by the postfix _ProfibusDP, _ProfinetIO and _SercosIII. The libraries are available in the library repository from IndraWorks version 14V16. The required library has to be added to the library manager of the standard application.



Only add libraries to the library manager that are actually required. This decreases the loading time of the IndraWorks project.

The libraries contains acknowledgement function blocks for Safety I/O modules of Bosch Rexroth operated at SafeLogic. The acknowledgement function blocks are used in the standard application to read out and acknowledge diagnostics of the Safety I/O modules of Bosch Rexroth. In order for the channel/the module to resume operation after troubleshooting has been completed, the user has to acknowledge this. The acknowledgement procedure depends on the manufacturer as well as on the field bus and the local bus of the I/O station.

The following acknowledgement function blocks can be provided in the field bus-specific libraries:

Profibus library: RIH_Safety_ProfibusDP

- IH_PBDPAck_PSDx_Inline Acknowledgement function block for Profibus Inline Safety I/O module on PROFIsafe basis
- IH_PBDPAck_PSDx_S20 Acknowledgement function block for Profibus S20 Safety I/O modules on PROFIsafe basis

Profinet library: RIH_Safety_ProfinetIO

- IH_PNIOAck_PSDx_Inline Acknowledgement function block for Profinet Inline Safety I/O modules on PROFIsafe basis
- IH_PNIOAck_PSDx_S20 Acknowledgement function block for Profinet S20 Safety I/O modules on PROFIsafe basis

Sercos library: RIH_Safety_SercosIII

- IH_SIIIAck_SSDx_S20 acknowledgement function block Sercos S20 Safety I/O modules on Csos basis

The function blocks use data types of theRIH_Safety library.

16.1 IH_PBDPAck_PSDx_Inline

Brief description

The **IH_PBDPAck_PSDx_Inline** function block retrieves and acknowledges diagnostics of PSDx modules (PROFIsafe Inline module) at the Profibus Inline bus coupler (R-IL PB BK DI8 DO4/CN).

The function block is enabled at a rising edge at "Execute" and returns with TRUE to the "Done" output after successful execution.

The function block can then be enabled again at a rising edge at "Execute". The user specifies how often the function block is enabled.

The "Active" output indicates that diagnostics were retrieved and that they are available.

The "Done" output is set when either all diagnostics are acknowledged or if they do not have to be acknowledged or they are standard diagnostics. The "Done" outputs is also set if there was no diagnostics.



The **IH_PBDPAck_PSDx_Inline** function block requires at least the **Profibus Inline bus coupler R-IL PB BK DI8 DO4/CN with the change index AD1 or higher**!

Target system	Library/folder
MLC	RIH_Safety_ProfibusDP
MTX	RIH_Safety_ProfibusDP

Tab. 16-1: Reference table

Interface

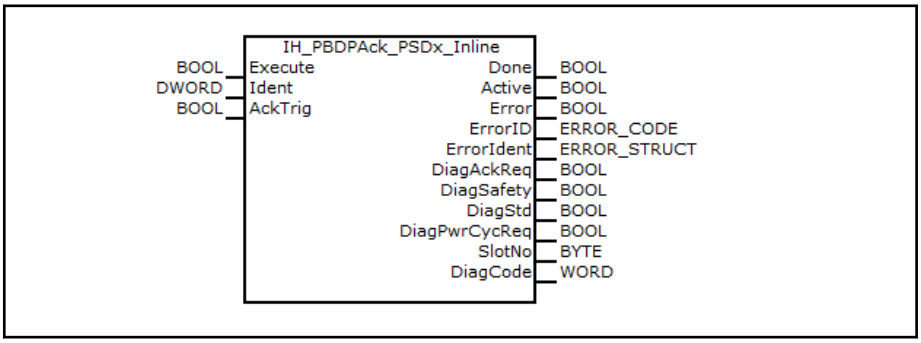


Fig. 16-1: FB IH_PBDPAck_PSDx_Inline interface description

I/O type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Starts the function at a rising edge
	Ident	DWORD	Identifier of the DP slave R-IL PB BK DI8 DO4/CN (Profibus Ident handle of the function IL_DPIdent, page 39) Identifier of the DP slave R-IL PB BK DI8 DO4/CN (Profibus Ident handle of the function IL_DPIdent) Note: Use "0" as slot!
	AckTrig	BOOL	Acknowledgement input: Rising edge required if "DiagAck-Req" is present

I/O type	Name	Type	Description
VAR_OUTPUT	Done	BOOL	TRUE indicates that either all diagnostics are acknowledged or that they do not have to be acknowledged or that they are standard diagnostics or that there was no diagnostics
	Active	BOOL	TRUE indicates that the function block is active
	Error	BOOL	TRUE indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Also refer to "Troubleshooting" on page 699
	DiagAckReq	BOOL	TRUE indicates that there is a diagnostics that has to be acknowledged and confirmed via "AckTrig"
	DiagSafety	BOOL	TRUE indicates a safety-relevant diagnostics
	DiagStd	BOOL	TRUE indicates a standard diagnostics
	DiagPwrCycReq	BOOL	TRUE indicates that a restart (power cycle) is required to ensure that the PSDx modules run with the "Power On" self test. Note: Perform the PowerCycle at the bus coupler. Additional voltage infeed terminals might be required
	SlotNo	BYTE	Slot number of the diagnostic-reporting module
	DiagCode	WORD	Diagnostic code

Tab. 16-2: I/O interface of the IH_PBDPAck_PSDx_Inline function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
AckTrig	BOOL			FALSE	Continuous if "DiagAckReq" is present
Ident	DWORD		16777214	0	Rising edge at "Execute"

Tab. 16-3: Min., max. and default values of the IH_PBDPAck_PSDx_Inline inputs

Types and constants

The types and constants facilitate detailed diagnostic information and provide diagnostic texts to the user.

The "IH_DIAGCODE_NO_EXTENDED_DIAG_AVAILABLE" constant contains the diagnostic code in case no extended diagnostics is available.

The "IH_DIAGCODE_PROFIBUS_SLAVE_NOT_AVAILABLE" constant contains the diagnostic code for the Profibus error “Device does not respond”.

The "IH_SAFETY_IO_DIAG_TEXT" structure contains diagnostic text and diagnostic code.

Name	Type	Comment
DiagCode	WORD	Diagnostic code
Text	STRING(255)	Diagnostic text

Tab. 16-4: "IH_SAFETY_IO_DIAG_TEXT"

The "SafetyDiagTexts" constant contains two arrays of type "IH_SAFETY_IO_DIAG_TEXT" with diagnostic texts for the INLINE-PSDx module:

- German: arInlinePnbSafetyDiagTexts_De
- English: arInlinePbSafetyDiagTexts_En

The "IH_PNIO_INLINE_SAFETY_DIAG_TEXT_AMOUNT" constant indicates the number of diagnostic texts.

Search in "arInlinePbSafetyDiagTexts_xx" for the "DiagCode" to determine the corresponding diagnostic text.

```
//Get DE texts from global text array
IF FB_PBDPAck_PSDx_Inline.DiagCode <>0 THEN
    FOR i:=1 TO IH_PBDP_INLINE_SAFETY_DIAG_TEXT_AMOUNT DO
        IF arInlinePbSafetyDiagTexts_De[i].DiagCode = FB_PBDPAck_PSDx_Inline.DiagCode THEN
            szString := concat (szString , arInlinePbSafetyDiagTexts_De[i].Text);
            EXIT;  //Text found
        END_IF
    END_FOR
END_IF
```

Fig. 16-2: Example to determine the diagnostic texts from "arInlinePbSafetyDiagTexts_xx"

Functional description

The **IH_PBDPAck_PSDx_Inline** function block detects and acknowledges diagnostics of PSDx modules at a Profibus Inline bus coupler (R-IL PB BK DI8 DO4/CN) using DPV1 read/write services.

The **IH_PBDPAck_PSDx_Inline** function block is based on the basic function block **IL_ExecuteDoneBase**.

There are different DPV1 function blocks in the function block, such as IL_DPReadDiag, page 43, IL_DPV1Read, page 40, IL_DPV1Write, page 41.

The "IH_PBDPAck_PSDx_Inline" function block has to be used by specifying the Profibus address of the Profibus Inline bus coupler and slot "0". The function block handles all modules located at the corresponding Profibus Inline bus coupler.



To keep the communication load on the PROFIBUS low, use as few "IH_PBDPAck_PSDx_Inline" function blocks as possible simultaneously.

If errors that have to be acknowledged are reported in multiple channels and modules, these errors are handled subsequently in the function block. Diagnostics have to be acknowledged individually. The diagnostic codes are output and separate edges are then required at the "AckTrig" input. TRUE at the "Done" output indicates that all diagnostics were acknowledged.

Acknowledging diagnostics is supported for the following PROFIsafe Inline modules:

- R-IB IL 24 PSDI 16
- R-IB IL 24 PSDI 8
- R-IB IL 24 PSDO 8
- R-IB IL 24 PSDO 4/4
- R-IB IL 24 PSDOR 4

In case of Safety diagnostics subject to acknowledgements, these are mostly channel-specific diagnostics, such as "Symmetry violation at input x channel y". After troubleshooting, these diagnostics have to be acknowledged before the channel/module is operated again.

⚠ WARNING **Observe the documentation of the PSDx modules!**

For the meaning of device-specific diagnostics and error codes of the Profibus Inline bus coupler (R-IL PB BK DI8 DO4/CN) and the PROFIsafe Inline modules, refer to the respective device documentation. Notes on troubleshooting as well as instructions on error elimination and acknowledgement are described in the respective documentation.

⚠ WARNING **Acknowledging an error can cause a dangerous state**

Apart from the specified exceptions, acknowledging an error causes an immediate return of the safe output/input to the operating state. Thus, ensure that the acknowledgement does not cause a dangerous machine state before acknowledging an error! When planning the machine/system, take into account that acknowledging may only be possible if the dangerous area is visible.

Acknowledging Safety-relevant diagnostics is initiated at a rising edge at "Execute".

First, **IL_DPReadDiag** is implicitly executed to detect, whether there is a diagnostics at the bus coupler or at it modules.

Based on the diagnostic feedback, the diagnostic class is determined. This diagnostic class provides information on whether it is a standard or a Safety diagnostics. Thus, the outputs "DiagSafety", "DiagStd" and "DiagPwrCycReq" are set.

The "DiagAckReq" output indicates whether a Safety diagnostics subject to acknowledgement is present and whether this diagnostics has to be acknowledged at a positive edge at the "AckTrig" after error elimination.

If all diagnostics subject to acknowledgement are processed, the "Done" output is set to TRUE. The "DiagCode" output shows the diagnostics "Finished".



The acknowledgment in the PSDx module requires several seconds.

Diagnostics

The "DiagCode" output returns the state of the Profibus Inline bus coupler and the Inline modules.

In addition, the "SlotNo" output indicates the diagnostic-reporting slot or the Inline module number.

Troubleshooting

The **IH_PBDPAck_PSDx_Inline** function block forwards the error codes of the DPV1 function blocks used at the own "Error" output (refer to the table **PB_DP_TABLE (16#0130)** Profibus DP errors).

The **IH_PBDPAck_PSDx_Inline** function block generates the following error codes:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#000000F9	0	Error in state machine when reading the confirmation for the acknowledgement
STATE_MACHINE_ERROR	16#000000FA	0	Error in state machine while writing to update the diagnostic buffer
INPUT_RANGE_ERROR	16#000000FB	0	Wrong slot: Use the "IL DPIdent" function with slot "0"

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#000000FC	0	Wrong diagnostic format selected in R-IL PB BK DI8 DO4/CN. Configure the format "Status PDU"
DEVICE_ERROR	16#000000FE	0	Device not supported. The bus coupler R-IL PB BK DI8 DO4/CN has to be used
STATE_MACHINE_ERROR	16#000000FF	0	Error in the state machine. If this error is output, the function block only generated this error

Tab. 16-5: Error codes generated by IH_PBDPAck_PSDx_Inline

16.2 IH_PBDPAck_PSDx_S20

Brief description

The **IH_PBDPAck_PSDx_S20** function block is used to read out and acknowledge diagnostics of the S20-PSDx modules (PROFIsafe modules) at the Profibus S20 bus coupler (S20-PB-BK).

The function block is enabled at a rising edge at the "Execute" input and returns with TRUE to the "Done" output after successful execution.

The function block can then be enabled again at a rising edge at the "Execute" input. The user specifies how often the function block is enabled.

The "Active" output displays if the function block is in operation, e.g. calling diagnostics.

The "Done" output is set when either all diagnostics are acknowledged or if they do not have to be acknowledged or they are standard diagnostics. The "Done" outputs is also set if there was no diagnostics.

☞ The function block IH_PBDPAck_PSDx_S20 requires the **Profibus S20 bus coupler S20-PB-BK** (R911173247) in edition AB1 and later.

Target system	Library/folder
MLC	RIH_Safety_ProfibusDP
MTX	RIH_Safety_ProfibusDP

Tab. 16-6: Reference table

Interface

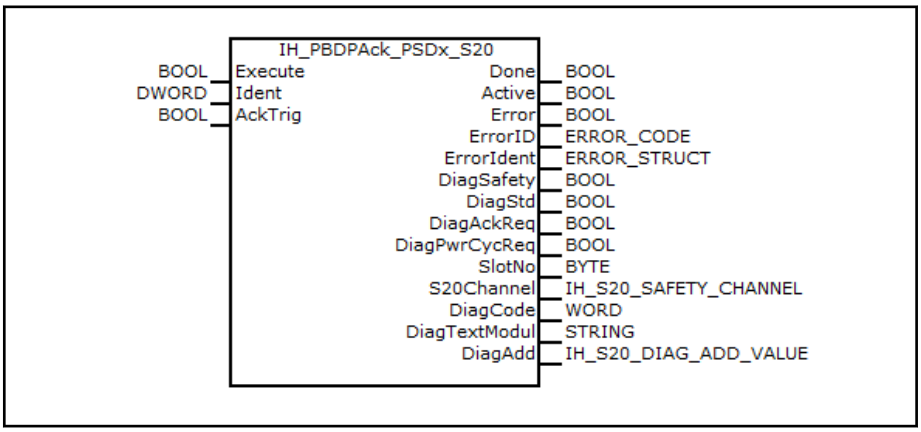


Fig. 16-3: Interface description of the IH_PBDPAck_PSDx_S20 function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Starts the function at a rising edge
	Ident	DWORD	Identifier of the DP slave "S20-PB-BK" (Profibus Ident handle of the function IL_DPIdent, page 39) Identifier of the DP slave "S20-PB-BK" (Profibus Ident handle of the function IL_DPIdent) **Note:** Use "0" as slot!
	AckTrig	BOOL	Acknowledgement input: Rising edge required if "DiagAckReq" is present
VAR_OUTPUT	Done	BOOL	TRUE indicates that either all diagnostics are acknowledged or that they do not have to be acknowledged or that they are standard diagnostics or that there was no diagnostics
	Active	BOOL	TRUE indicates that the function block is active
	Error	BOOL	TRUE indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information See also Troubleshooting, page 705
	DiagSafety	BOOL	TRUE indicates a Safety module diagnostics (Safety diagnostics)
	DiagStd	BOOL	TRUE indicates a standard diagnostics.
	DiagAckReq	BOOL	TRUE indicates that there is a diagnostics that has to be acknowledged and confirmed via "AckTrig"
	DiagPwrCycReq	BOOL	TRUE indicates that a restart/PowerCycle (switch power off/on) is required to ensure that the S20-PSDx modules run with the "Power On" self test. **Note:** Perform the PowerCycle at the bus coupler. Additional voltage infeed terminals might be required
	SlotNo	BYTE	Slot number of the diagnostic-reporting module
	S20Channel	IH_S20_SAFETY_CHANNEL	Channel for which the diagnostics is pending. Value 255 signifies that the entire module is affected. **Note:** SafetyIoNo, SafetyChannelNo, SafetyIoChannelName are only valid, if "DiagSafety" is TRUE.
	DiagCode	WORD	S20 diagnostic code
	DiagTextModul	STRING	S20 diagnostic text directly by the module, max. 50 characters English is default language
	DiagAdd	IH_S20_DIAG_ADD_VALUE	S20 additional diagnostics (code and text)

Tab. 16-7: Inputs/outputs of the IH_PBDPAck_PSDx_S20 function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Ident	DWORD		16777214	0	Rising edge at "Execute"
AckTrig	BOOL			FALSE	Continuous if "DiagAckReq" is reported

Tab. 16-8: Min./max. and default values of the function block inputs IH_PBDPAck_PSDx_S20

Types and constants

The types and constants facilitate detailed diagnostic information and provide diagnostic texts to the user.

The "IH_S20_DIAG_ADD_VALUE" structure contains the additional diagnostic code and the selected texts.

Name	Type	Comment
Value	DWORD	S20 Additional diagnostic value
Text	STRING	Text for the additional diagnostic value

Tab. 16-9: "IH_S20_DIAG_ADD_VALUE"

The "IH_S20_SAFETY_CHANNEL" structure contains the channel and the input assignment.

Name	Type	Value range
S20ChannelNo	BYTE	0 - 7, 255 (255: whole module is affected)
SafetyIoNo	BYTE	0 - 3, 255 (255: whole module is affected)
SafetyChannelNo	BYTE	1, 2, 255 (255: whole module is affected)
SafetyIoChannelName	STRING	'OUTx_CHy', 'INx_CHy', ''

Tab. 16-10: "IH_S20_SAFETY_CHANNEL"

The "IH_SAFETY_IO_DIAG_TEXT" structure contains diagnostic text and diagnostic code.

Name	Type	Comment
DiagCode	WORD	Diagnostic code
Text	STRING(255)	Diagnostic text

Tab. 16-11: "IH_SAFETY_IO_DIAG_TEXT"

The "IH_DIAGCODE_NO_EXTENDED_DIAG_AVAILABLE" constant contains the diagnostic code in case no extended diagnostics is available.

The "IH_DIAGCODE_PROFIBUS_SLAVE_NOT_AVAILABLE" constant contains the diagnostic code for the Profibus error “Device does not respond”.

The "SafetyDiagTexts" constant contains two arrays of type "IH_SAFETY_IO_DIAG_TEXT" with diagnostic texts for the S20-PSDx module:

- German: arS20PbSafetyDiagTexts_De
- English: arS20PbSafetyDiagTexts_En

The "IH_PBDP_S20_SAFETY_DIAG_TEXT_AMOUNT" constant indicates the number of diagnostic texts.

Search in "arS20PbSafetyDiagTexts_xx" for the "DiagCode" to determine the corresponding diagnostic text.

Example to determine the diagnostic texts from "arS20PbSafetyDiagTexts_xx"

Program:

```
//Get DE diagnosis text from global text array
IF fbIH_PBDPAck_PSDx_S20.DiagCode <> 0 THEN
  FOR i:=1 TO IH_PBDP_S20_SAFETY_DIAG_TEXT_AMOUNT DO
    IF arS20PbSafetyDiagTexts_De[i].DiagCode = fbIH_PBDPAck_PSDx_S20.DiagCode THEN
      szStringS20 := concat (szStringS20 , arS20PbSafetyDiagTexts_De[i].Text);
      bTextFoundS20 := TRUE;
      EXIT;  //Text found
    END_IF
  END_FOR
END_IF
IF bTextFoundS20 = FALSE THEN
  // If no text found use text from modul
  szStringS20 := concat (szStringS20 , fbIH_PBDPAck_PSDx_S20.DiagTextModul);
END_IF
```

Functional description

The "IH_PBDPAck_PSDx_S20" function block is used to read out and acknowledge diagnostics of the S20-PSDx modules (PROFIsafe modules) at the Profibus S20 bus coupler (S20-PB-BK).

The "IH_PBDPAck_PSDx_S20" function block is based on the basic function block "IL_ExecuteDoneBase".

There are different DPV1 function blocks in the function block, such as IL_DPReadDiag, page 43, IL_DPV1Read, page 40, IL_DPV1Write, page 41.

The "IH_PBDPAck_PSDx_S20" function block has to be used by specifying the Profibus address of the Profibus S20 bus coupler and slot "0". The function block contains all S20-PSDx modules located at the Profibus S20 bus coupler.



To keep the communication load on the Profibus low, use as few "IH_PBDPAck_PSDx_S20" function blocks as possible simultaneously.

If errors that have to be acknowledged are reported in multiple channels and multiple modules, they are handled subsequently in the function block. Diagnostics have to be acknowledged individually. This means, the diagnostic code is output and a separate edge is required at the "AckTrig" input. TRUE at the "Done" output and FALSE at the "DiagAckReq" output indicates that all diagnostics were acknowledged.

Acknowledging diagnostics is supported for the following PROFIsafe S20-PSDx modules:

- S20-PSDI-8/4
- S20-PSDO-8/3

In case of Safety diagnostics subject to acknowledgements, these are mostly channel-specific diagnostics, such as "Symmetry violation at input x channel y". After troubleshooting, these diagnostics have to be acknowledged before the channel/module can be operated again.

⚠ WARNING **Refer to the S20-PSDx module documentation**

For the meaning of device-specific diagnostics and error codes of the Profibus S20 bus coupler (S20-PB-BK) and the PROFIsafe S20 modules, refer to the respective device documentation. Notes on troubleshooting as well as instructions on error elimination and acknowledgement are described in the respective documentation.

⚠ WARNING **Acknowledgement can cause a dangerous state**

Apart from the specified exceptions, acknowledging an error causes an immediate return of the safe output/safe input to the operating state. Thus, ensure that the acknowledgement does not cause a dangerous machine state before acknowledging an error! When planning the machine/system, take into account that acknowledging may only be possible if the dangerous area is visible.

Acknowledging and reading out Safety-relevant diagnostics is initiated at a rising edge at the "Execute" input.

Initially, the command configuration is read out once at S20-PB-BK. Thus, it is known which slots contain the S20-PSDx modules. If the “Ident” input parameters (master, segment or station) change at function block runtime, the command configuration is read out again at the S20-PB-BK.

First, **IL_DPReadDiag** is implicitly executed to detect, whether there is a diagnostics at the bus coupler or at it modules.

The function block searches for the first S20-PSDx module with a diagnostics. (Search starts at the lowest slot number).

If the Safety diagnostics has been acknowledged successfully, the next diagnostics is handled by this module (usually by a different channel).

If there are no diagnostics at this module anymore, the next S20-PSDx module is searched which has diagnostics.

If there are no S20-PSDx modules with diagnostics, the first standard module is handled which has diagnostics.

Thus, the outputs "DiagSafety", "DiagStd", "DiagAckReq", "DiagPwrCycReq", "SlotNo", "S20Channel" und "DiagCode" are set on this basis.

The "DiagAckReq" output indicates whether a Safety diagnostics subject to acknowledgement is present and whether this diagnostics has to be acknowledged at a positive edge at the "AckTrig" after error elimination. The "ResetDiag object 0x0019" is internally used for acknowledging.

If all diagnostics subject to acknowledgement are processed, the "Done" output is set to TRUE.

Diagnostics

The "DiagCode" output returns the state of the Profibus S20 bus coupler and the S20 modules.

The "SlotNo" output indicates the diagnostic-reporting slot or the S20 module number.

The "S20Channel" output shows channel reporting the diagnostics. Value 255 signifies that the entire module is affected. The basis is the diagnostic object "DiagState object 0x0018, index 3: Channel/Group/Modules".

The "DiagTextModul" output returns the diagnostic text directly from the module. This corresponds to the diagnostic object "DiagState object 0x0018, index 5: Text".

The "DiagAdd" output returns additional diagnostics (code and text). The diagnostic object "DiagStateAddValue object 0x0034" is read out.

Troubleshooting

The "IH_PBDPAck_PSDx_S20" function block forwards the error codes of the internally used function blocks at the individual error output. Refer to the table **PB_DP_TABLE (16#0130)** Profibus DP error.

The IH_PBDPAck_PSDx_S20 function block generates the following error codes:

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#000000FC	0	Wrong diagnostic format selected in S20-PB-BK. Please configure the format "Common Format"
INPUT_RANGE_ERROR	16#000000FB	0	Wrong slot: Use the "IL DPIdent" function with slot "0"
DEVICE_ERROR	16#000000FD	0	The maximum number of 20 subsequent acknowledgements for the same slot is reached
DEVICE_ERROR	16#000000FE	0	Device not supported. Use bus coupler S20-PB-BK
STATE_MACHINE_ERROR	16#000000FF	0	Error in the state machine. If this error is output, the function block only generated this error

Tab. 16-12: Error codes generated by IH_PBDPAck_PSDx_S20

17 RIH_Safety_ProfinetIO.library

Introduction and overview: RIH_Safety_ProfibusDP, RIH_Safety_ProfinetIO and RIH_Safety_SercosIII

Field bus-specific functionalities of SafeLogic for use in the standard application are bundled in these libraries. The field bus assignment is identified by the postfix _ProfibusDP, _ProfinetIO and _SercosIII. The libraries are available in the library repository from IndraWorks version 14V16. The required library has to be added to the library manager of the standard application.



Only add libraries to the library manager that are actually required. This decreases the loading time of the IndraWorks project.

The libraries contains acknowledgement function blocks for Safety I/O modules of Bosch Rexroth operated at SafeLogic. The acknowledgement function blocks are used in the standard application to read out and acknowledge diagnostics of the Safety I/O modules of Bosch Rexroth. In order for the channel/the module to resume operation after troubleshooting has been completed, the user has to acknowledge this. The acknowledgement procedure depends on the manufacturer as well as on the field bus and the local bus of the I/O station.

The following acknowledgement function blocks can be provided in the field bus-specific libraries:

Profibus library: RIH_Safety_ProfibusDP

- IH_PBDPAck_PSDx_Inline Acknowledgement function block for Profibus Inline Safety I/O module on PROFIsafe basis
- IH_PBDPAck_PSDx_S20 Acknowledgement function block for Profibus S20 Safety I/O modules on PROFIsafe basis

Profinet library: RIH_Safety_ProfinetIO

- IH_PNIOAck_PSDx_Inline Acknowledgement function block for Profinet Inline Safety I/O modules on PROFIsafe basis
- IH_PNIOAck_PSDx_S20 Acknowledgement function block for Profinet S20 Safety I/O modules on PROFIsafe basis

Sercos library: RIH_Safety_SercosIII

- IH_SIIIAck_SSDx_S20 acknowledgement function block Sercos S20 Safety I/O modules on Csos basis

The function blocks use data types of theRIH_Safety library.

17.1 IH_PNIOAck_PSDx_S20

Brief description

The **IH_PNIOAck_PSDx_S20** function block is used to read out and acknowledge diagnostics of the S20-PSDx modules (PROFIsafe modules) at the PROFINET S20 bus coupler (S20-PN-BK+).

The function block is enabled at a rising edge at the "Execute" input and returns with TRUE to the "Done" output after successful execution.

The function block can then be enabled again at a rising edge at the "Execute" input. The user specifies how often the function block is enabled.

The "Active" output displays if the function block is in operation, e.g. calling diagnostics.

The "Done" output is set when either all diagnostics are acknowledged or if they do not have to be acknowledged or they are standard diagnostics. The "Done" outputs is also set if there was no diagnostics.



The function block IH_PNIOAck_PSDx_S20 requires the **PROFINET S20 bus coupler S20-PN-BK+** (R911173359).

Target system	Library/folder
MLC	RIH_Safety_ProfinetIO
MTX	RIH_Safety_ProfinetIO

Tab. 17-1: Reference table

Interface

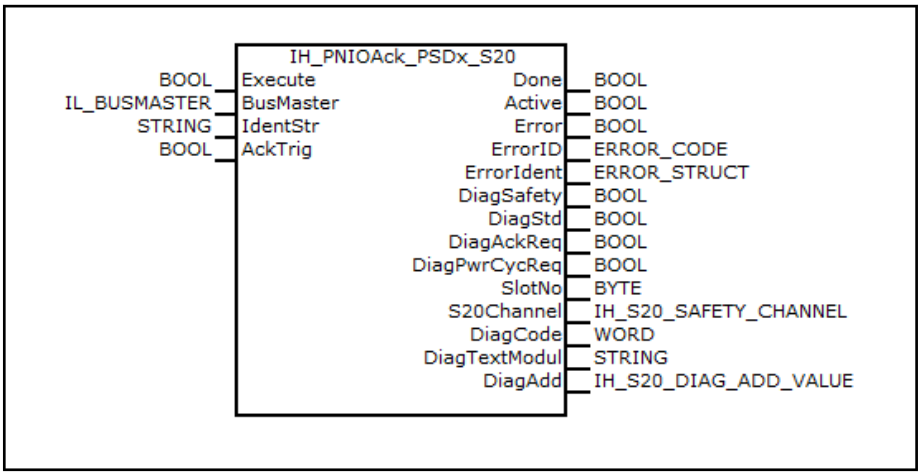


Fig. 17-1: Interface description of the IH_PNIOAck_PSDx_S20 function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Starts the function at a rising edge
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string (station name) of PN device S20-PN-BK +
	AckTrig	BOOL	Acknowledgement input: Rising edge required if "DiagAck-Req" is present
VAR_OUTPUT	Done	BOOL	TRUE indicates that either all diagnostics are acknowledged or if they do not have to be acknowledged or they are standard diagnostics.
	Active	BOOL	TRUE indicates that the function block is active
	Error	BOOL	TRUE indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information See also Troubleshooting, page 712
	DiagSafety	BOOL	TRUE indicates a Safety module diagnostics (Safety diagnostics)
	DiagStd	BOOL	TRUE indicates a standard diagnostics

	Name	Type	Description
	DiagAckReq	BOOL	TRUE indicates that there is a diagnostics that has to be acknowledged and confirmed via "AckTrig".
	DiagPwrCycReq	BOOL	TRUE indicates that a restart/power cycle (switch power off/on) is required to ensure that the S20-PSDx modules run with the "Power On" self test. **Note:** Perform the PowerCycle at the bus coupler. Additional voltage infeed terminals might be required
	SlotNo	BYTE	Slot number of the diagnostic-reporting module
	S20Channel	IH_S20_SAFETY_CHANNEL	Channel for which the diagnostics is pending. Value 255 signifies that the entire module is affected. **Note:** SafetyIoNo, SafetyChannelNo, SafetyIoChannelName are only valid, if "DiagSafety" is TRUE.
	DiagCode	WORD	S20 diagnostic code
	DiagTextModul	STRING	S20 diagnostic text directly by the module, max. 50 characters English is default language
	DiagAdd	IH_S20_DIAG_ADD_VALUE	S20 additional diagnostics (code and text)

Tab. 17-2: Inputs/outputs of the IH_PNIOAck_PSDx_S20 function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4		Rising edge at "Execute"
IdentStr	STRING			''	Rising edge at "Execute"
AckTrig	BOOL			FALSE	Continuous if "DiagAckReq" is reported

Tab. 17-3: Min./max. and default values of the function block inputs IH_PNIOAck_PSDx_S20

Types and constants

The types and constants facilitate detailed diagnostic information and provide diagnostic texts to the user.

The "IH_S20_DIAG_ADD_VALUE" structure contains the additional diagnostic code and the selected texts.

Name	Type	Comment
Value	DWORD	S20 Additional diagnostic value
Text	STRING	Text for the additional diagnostic value

Tab. 17-4: "IH_S20_DIAG_ADD_VALUE"

The "IH_S20_SAFETY_CHANNEL" structure contains the channel and the input assignment.

Name	Type	Value range
S20ChannelNo	BYTE	0 - 7, 255 (255: whole module is affected)
SafetyIoNo	BYTE	0 - 3, 255 (255: whole module is affected)
SafetyChannelNo	BYTE	1, 2, 255 (255: whole module is affected)
SafetyIoChannelName	STRING	'OUTx_CHy', 'INx_CHy', ''

Tab. 17-5: "IH_S20_SAFETY_CHANNEL"

The "IH_SAFETY_IO_DIAG_TEXT" structure contains diagnostic text and diagnostic code.

Name	Type	Comment
DiagCode	WORD	Diagnostic code
Text	STRING(255)	Diagnostic text

Tab. 17-6: "IH_SAFETY_IO_DIAG_TEXT"

The "IH_DIAGCODE_MISSING_MODULE" constant contains the diagnostic code for the PROFINET error “Module not available”.

The "IH_DIAGCODE_WRONG_MODULE" constant contains the diagnostic code for the PROFINET error “Wrong module”.

The "SafetyDiagTexts" constant contains two arrays of type "IH_SAFETY_IO_DIAG_TEXT" with diagnostic texts for the S20-PSDx module:

- German: arS20PnSafetyDiagTexts_De
- English: arS20PnSafetyDiagTexts_En

The "IH_PNIO_S20_SAFETY_DIAG_TEXT_AMOUNT" constant indicates the number of diagnostic texts.

Search in "arS20PnSafetyDiagTexts_xx" for the "DiagCode" to determine the corresponding diagnostic text.

```
//Get DE diagnosis text from global text array
IF FB_PNIOAck_PSDx_S20.DiagCode <> 0 THEN
    FOR i:=1 TO IH_PNIO_S20_SAFETY_DIAG_TEXT_AMOUNT DO
        IF arS20PnSafetyDiagTexts_De[i].DiagCode = FB_PNIOAck_PSDx_S20.DiagCode THEN
            szStringS20 := concat (szStringS20 , arS20PnSafetyDiagTexts_De[i].Text);
            bTextFoundS20 := TRUE;
            EXIT;  //Text found
        END_IF
    END_FOR
END_IF
IF bTextFoundS20 = FALSE THEN
    // If no text found use text from modul
    szStringS20 := concat (szStringS20 , FB_PNIOAck_PSDx_S20.DiagTextModul);
END_IF
```

Fig. 17-2: Example to determine the diagnostic texts from "arS20PnSafetyDiagTexts_xx"

Functional description

The "IH_PNIOAck_PSDx_S20" function block is used to read out and acknowledge diagnostics of the S20-PSDx modules (PROFIsafe modules) at the PROFINET S20 bus coupler (S20-PN-BK+).

The "IH_PNIOAck_PSDx_S20" function block is based on the basic function block "IL_ExecuteDoneBase".

Different function blocks are contained in the function block: 128, page 128, 71, page 71 und 79, page 79.

The “IH_PNIOAck_PSDx_S20” function block has to be used by specifying the "station name" and "PNIO controller" of the PROFINET S20 bus coupler. The function block contains all S20-PSDx modules located at the PROFINET S20 bus coupler.

To keep the communication load on the PROFINET low, use as few "IH_PNIOAck_PSDx_S20" function blocks as possible simultaneously.

If errors that have to be acknowledged are reported in multiple channels and multiple modules, they are handled subsequently in the function block. Diagnostics have to be acknowledged individually. This means, the diagnostic code is output and a separate edge is required at the "AckTrig" input. TRUE at the "Done" output and FALSE at the "DiagAckReq" output indicates that all diagnostics were acknowledged.

Acknowledging diagnostics is supported for the following PROFIsafe S20-PSDx modules:

- S20-PSDI-8/4
- S20-PSDO-8/3

In case of Safety diagnostics subject to acknowledgements, these are mostly channel-specific diagnostics, such as "Symmetry violation at input x channel y". After troubleshooting, these diagnostics have to be acknowledged before the channel/module can be operated again.

⚠ WARNING **Refer to the S20-PSDx module documentation**

For the meaning of device-specific diagnostics and error codes of the PROFINET S20 bus coupler (S20-PN-BK+) and the PROFIsafe S20 modules, refer to the respective device documentation. Notes on troubleshooting as well as instructions on error elimination and acknowledgement are described in the respective documentation.

⚠ WARNING **Acknowledgement can cause a dangerous state**

Apart from the specified exceptions, acknowledging an error causes an immediate return of the safe output/input to the operating state. Thus, ensure that the acknowledgement does not cause a dangerous machine state before acknowledging an error! When planning the machine/system, take into account that acknowledging may only be possible if the dangerous area is visible.

Acknowledging and reading out Safety-relevant diagnostics is initiated at a rising edge at the "Execute" input.

Initially, the command configuration at S20-PN-BK+ is read out once. Thus, it is known which slots contain the S20-PSDx modules. If the input parameters BusMaster or IdentStr (station name) change during the function block runtime, the command configuration at the S20-S3-BK+ is read out again.

First, the function block 128, page 128 is executed to detect, whether there is a diagnostics at the bus coupler or at it modules.

The function block searches for the first S20-PSDx module with a diagnostics. (Search starts at the lowest slot number).

If the Safety diagnostics has been acknowledged successfully, the next diagnostics is handled by this module (usually by a different channel).

If there are no diagnostics at this module anymore, the next S20-PSDx module is searched which has diagnostics.

If there are no S20-PSDx modules with diagnostics, the first standard module is handled which has diagnostics.

Thus, the outputs "DiagSafety", "DiagStd", "DiagAckReq", "DiagPwrCycReq", "SlotNo", "S20Channel" und "DiagCode" are set on this basis.

The "DiagAckReq" output indicates whether a Safety diagnostics subject to acknowledgement is present and whether this diagnostics has to be acknowledged at a positive edge at the "AckTrig" after error elimination. The "ResetDiag object 0x0019" is internally used for acknowledging.

If all diagnostics subject to acknowledgement are processed, the "Done" output is set to TRUE.

Diagnostics

The "DiagCode" output returns the state of the PROFINET S20 bus coupler and the S20 modules.

The "SlotNo" output indicates the diagnostic-reporting slot or the S20 module number.

The "S20Channel" output shows channel reporting the diagnostics. Value 255 signifies that the entire module is affected. The basis is the diagnostic object "DiagState object 0x0018, index 3: Channel/Group/Modules".

The "DiagTextModul" output returns the diagnostic text directly from the module. This corresponds to the diagnostic object "DiagState object 0x0018, index 5: Text".

The "DiagAdd" output returns additional diagnostics (code and text). The diagnostic object "DiagStateAddValue object 0x0034" is read out.

Troubleshooting

The " IH_PNIOAck_PSDx_S20" function block forwards the error codes of the internally used function blocks at the individual error output. See table **PROFINETIO_TABLE (16#0200)**.

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#00000003	16#C00C0056	Service not available. The device is not in the I/O data exchange.

Tab. 17-7: Example of error codes of the internally used function blocks

The IH_PNIOAck_PSDx_S20 function block generates the following error codes:

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#000000FD	0	The maximum number of 20 subsequent acknowledgements for the same slot is reached.
DEVICE_ERROR	16#000000FE	0	Device not supported. Use bus coupler S20-PN-BK+
STATE_MACHINE_ERROR	16#000000FF	0	Error in state machine. If this error is output, the function block only generated this error.

Tab. 17-8: Error codes generated by IH_PNIOAck_PSDx_S20

17.2 IH_PNIOAck_PSDx_Inline

Brief description

The **IH_PNIOAck_PSDx_Inline** function block reads out and acknowledges diagnostics of PROFIsafe INLINE module (PSDx) at the PROFINET INLINE bus coupler (R-IL PN BK DI8 DO4).

The function block is enabled at a rising edge at the "Execute" input and returns with TRUE to the "Done" output after successful execution.

The function block can then be enabled again at a rising edge at the "Execute" input. The user specifies how often the function block is enabled.

The "Active" output displays if the function block is in operation, e.g. calling diagnostics.

The "Done" output is set when either all diagnostics are acknowledged or if they do not have to be acknowledged or they are standard diagnostics. The "Done" outputs is also set if there was no diagnostics.

☞ The IH_PNIOAck_PSDx_Inline function block requires the **PROFINET INLINE bus coupler R-IL PN BK DI8 DO4** with change index AD1 (FWA version 3.41) or higher.

Target system	Library/folder
MLC	RIH_Safety_ProfinetIO
MTX	RIH_Safety_ProfinetIO

Tab. 17-9: Reference table

Interface

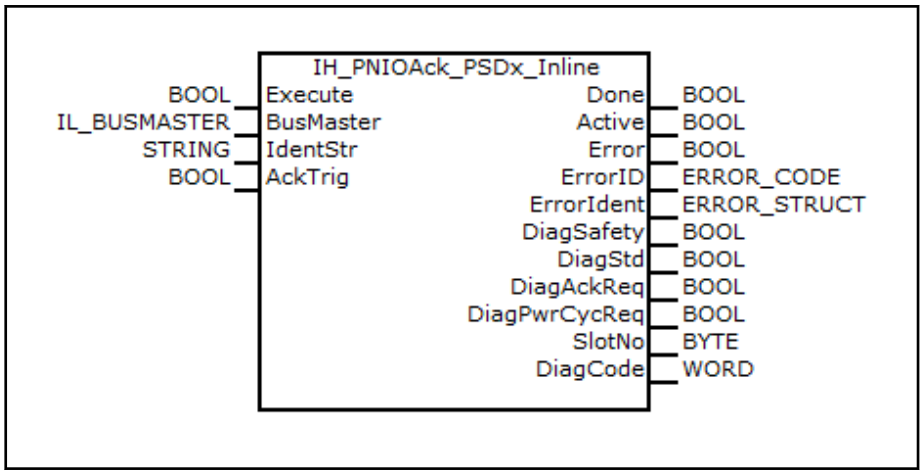


Fig. 17-3: Interface description of the IIH_PNIOAck_PSDx_Inline function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Starts the function at a rising edge
	BusMaster	IL_BUSMASTER	Instance of the Profinet I/O controller in the order of the configuration tree. If only one PROFINET IO controller is available, the value of this controller is always IL_BUSMASTER_0. Also refer to IL_BUSMASTER on page 184
	IdentStr	STRING	Identification string (station name) of PN device R-IL PN BK DI8 DO4
	AckTrig	BOOL	Acknowledgement input: Rising edge required if "DiagAckReq" is present

	Name	Type	Description
VAR_OUTPUT	Done	BOOL	TRUE indicates that either all diagnostics are acknowledged or if they do not have to be acknowledged or they are standard diagnostics.
	Active	BOOL	TRUE indicates that the function block is active
	Error	BOOL	TRUE indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information See also Troubleshooting, page 717
	DiagSafety	BOOL	TRUE indicates a Safety module diagnostics (Safety diagnostics)
	DiagStd	BOOL	TRUE indicates a standard diagnostics
	DiagAckReq	BOOL	TRUE indicates that there is a diagnostics that has to be acknowledged and confirmed via "AckTrig".
	DiagPwrCycReq	BOOL	TRUE indicates that a restart/power cycle (switch power off/on) is required to ensure that the PROFIsafe INLINE modules run with the "Power On" self test. **Note:** Perform the PowerCycle at the bus coupler. Additional voltage infeed terminals might be required
	SlotNo	BYTE	Slot number of the diagnostic-reporting module
	DiagCode	WORD	Diagnostic code

Tab. 17-10: Inputs/outputs of the IH_PNIOAck_PSDx_Inline function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_4		Rising edge at "Execute"
IdentStr	STRING			''	Rising edge at "Execute"
AckTrig	BOOL			FALSE	Continuous if "DiagAckReq" is reported

Tab. 17-11: Min./max. and default values of the function block inputs IH_PNIOAck_PSDx_Inline

Types and constants

The types and constants facilitate detailed diagnostic information and provide diagnostic texts to the user.

The "IH_DIAGCODE_MISSING_MODULE" constant contains the diagnostic code for the PROFINET error “Module not available”.

The "IH_DIAGCODE_WRONG_MODULE" constant contains the diagnostic code for the PROFINET error “Wrong module”.

The "IH_SAFETY_IO_DIAG_TEXT" structure contains diagnostic text and diagnostic code.

Name	Type	Comment
DiagCode	WORD	Diagnostic code
Text	STRING(255)	Diagnostic text

Tab. 17-12: "IH_SAFETY_IO_DIAG_TEXT"

The "SafetyDiagTexts" constant contains two arrays of type "IH_SAFETY_IO_DIAG_TEXT" with diagnostic texts for the INLINE-PSDx module:

- German: arInlinePnSafetyDiagTexts_De
- English: arInlinePnSafetyDiagTexts_En

The "IH_PNIO_INLINE_SAFETY_DIAG_TEXT_AMOUNT" constant indicates the number of diagnostic texts.

Search in "arInlinePnSafetyDiagTexts_xx" for the "DiagCode" to determine the corresponding diagnostic text.

```
//Get DE texts from global text array
IF FB_PNIOAck_PSDx_Inline.DiagCode <>0 THEN
    FOR i:=1 TO IH_PNIO_INLINE_SAFETY_DIAG_TEXT_AMOUNT DO
        IF arInlinePnSafetyDiagTexts_De[i].DiagCode = FB_PNIOAck_PSDx_Inline.DiagCode THEN
            szString := concat (szString , arInlinePnSafetyDiagTexts_De[i].Text);
            EXIT;  //Text found
        END_IF
    END_FOR
END_IF
```

Fig. 17-4: Example to determine the diagnostic texts from "arInlinePnSafetyDiagTexts_xx"

Functional description

The "IH_PNIOAck_PSDx_Inline" function block reads out and acknowledges diagnostics of PROFIsafe INLINE module (PROFIsafe modules) at the PROFINET INLINE bus coupler (R-IL PN BK DI8 DO4).

The "IH_PNIOAck_PSDx_Inline" function block is based on the basic function block "IL_ExecuteDoneBase".

Different function blocks are contained in the function block: 128, page 128, 71, page 71 und 79, page 79.

The “IH_PNIOAck_PSDx_Inline” function block has to be used by specifying the "station name" and "PNIO controller" of the PROFINET INLINE bus coupler. The function block handles all PROFIsafe INLINE modules located at the corresponding Profinet Inline bus coupler.



To keep the communication load on the PROFINET low, use as few "IH_PNIOAck_PSDx_Inline" function blocks as possible simultaneously.

If errors that have to be acknowledged are reported in multiple channels and multiple modules, they are handled subsequently in the function block. Diagnostics have to be acknowledged individually. This means, the diagnostic code is output and a separate edge is required at the "AckTrig" input. TRUE at the "Done" output and FALSE at the "DiagAckReq" output indicates that all diagnostics were acknowledged.

Acknowledging diagnostics is supported for the following PROFIsafe Inline modules:

- R-IB IL 24 PSDI 16
- R-IB IL 24 PSDI 8
- R-IB IL 24 PSDO 8

- R-IB IL 24 PSDO 4/4
- R-IB IL 24 PSDOR 4

In case of Safety diagnostics subject to acknowledgements, these are mostly channel-specific diagnostics, such as "Symmetry violation at input x channel y". After troubleshooting, these diagnostics have to be acknowledged before the channel/module can be operated again.

⚠ WARNING **Refer to the PROFIsafe INLINE module documentation**

For the meaning of device-specific diagnostics and error codes of the PROFINET INLINE bus coupler (R-IL PN BK DI8 DO4) and the PROFIsafe INLINE modules, refer to the respective device documentation. Notes on troubleshooting as well as instructions on error elimination and acknowledgement are described in the respective documentation.

⚠ WARNING **Acknowledgement can cause a dangerous state**

Apart from the specified exceptions, acknowledging an error causes an immediate return of the safe output/input to the operating state. Thus, ensure that the acknowledgement does not cause a dangerous machine state before acknowledging an error! When planning the machine/system, take into account that acknowledging may only be possible if the dangerous area is visible.

Acknowledging and reading out Safety-relevant diagnostics is initiated at a rising edge at the "Execute" input.

Initially, the command configuration at R-IL PN BK DI8 DO4 is read out once. Thus, it is known which slots contain the PROFIsafe INLINE modules. If the input parameters BusMaster or IdentStr (station name) change during the function block runtime, the command configuration at the R-IL PN BK DI8 DO4 is read out again.

First, the function block 128, page 128 is executed to detect, whether there is a diagnostics at the bus coupler or at it modules.

The function block searches for the first INLINE PSDx module with a diagnostics. (Search starts at the lowest slot number).

If the Safety diagnostics has been acknowledged successfully, the next diagnostics is handled by this module (usually by a different channel).

If there are no diagnostics at this module anymore, the next INLINE-PSDx module is searched which has diagnostics.

If there are no PROFIsafe INLINE modules with diagnostics, the first standard module is handled which has diagnostics.

Thus, the outputs "DiagSafety", "DiagStd", "DiagAckReq", "DiagPwrCycReq", "SlotNo" and "DiagCode" are set on this basis.

The "DiagAckReq" output indicates whether a Safety diagnostics subject to acknowledgement is present and whether this diagnostics has to be acknowledged at a positive edge at the "AckTrig" after error elimination.

If all diagnostics subject to acknowledgement are processed, the "Done" output is set to TRUE.

Diagnostics The "DiagCode" output returns the state of the Profinet Inline bus coupler and the Inline modules.

The "SlotNo" output indicates the diagnostic-reporting slot or the Inline module number.

Troubleshooting

The " IH_PNIOAck_PSDx_Inline" function block forwards the error codes of the internally used function blocks at the individual error output. See table **PROFINETIO_TABLE (16#0200)**.

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#00000003	16#C00C0056	Service not available. The device is not in the I/O data exchange.

Tab. 17-13: Example of error codes of the internally used function blocks

The "IH_PNIOAck_PSDx_Inline" function block generates the following error codes:

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#000000FD	0	The maximum number of 20 subsequent acknowledgements for the same slot is reached.
DEVICE_ERROR	16#000000FE	0	Unsupported FWA version of the device; use bus coupler R-IL PN BK DI8 DO4 with FW version 3.41 or higher.
STATE_MACHINE_ERROR	16#000000FF	0	Error in state machine. If this error is output, the function block only generated this error.

Tab. 17-14: Error codes generated by "IH_PNIOAck_PSDx_Inline"

18 RIH_Safety_SercosIII.library

Introduction and overview: RIH_Safety_ProfibusDP, RIH_Safety_ProfinetIO and RIH_Safety_SercosIII

Field bus-specific functionalities of SafeLogic for use in the standard application are bundled in these libraries. The field bus assignment is identified by the postfix _ProfibusDP, _ProfinetIO and _SercosIII. The libraries are available in the library repository from IndraWorks version 14V16. The required library has to be added to the library manager of the standard application.



Only add libraries to the library manager that are actually required. This decreases the loading time of the IndraWorks project.

The libraries contains acknowledgement function blocks for Safety I/O modules of Bosch Rexroth operated at SafeLogic. The acknowledgement function blocks are used in the standard application to read out and acknowledge diagnostics of the Safety I/O modules of Bosch Rexroth. In order for the channel/the module to resume operation after troubleshooting has been completed, the user has to acknowledge this. The acknowledgement procedure depends on the manufacturer as well as on the field bus and the local bus of the I/O station.

The following acknowledgement function blocks can be provided in the field bus-specific libraries:

Profibus library: RIH_Safety_ProfibusDP

- IH_PBDPAck_PSDx_Inline Acknowledgement function block for Profibus Inline Safety I/O module on PROFIsafe basis
- IH_PBDPAck_PSDx_S20 Acknowledgement function block for Profibus S20 Safety I/O modules on PROFIsafe basis

Profinet library: RIH_Safety_ProfinetIO

- IH_PNIOAck_PSDx_Inline Acknowledgement function block for Profinet Inline Safety I/O modules on PROFIsafe basis
- IH_PNIOAck_PSDx_S20 Acknowledgement function block for Profinet S20 Safety I/O modules on PROFIsafe basis

Sercos library: RIH_Safety_SercosIII

- IH_SIIIAck_SSDx_S20 acknowledgement function block Sercos S20 Safety I/O modules on Csos basis

The function blocks use data types of theRIH_Safety library.

18.1 IH_SIIIAck_SSDx_S20

Brief description

The **IH_SIIIAck_SSDx_S20** function block is used to read out and acknowledge diagnostics of the S20-SSDx modules (Csos modules) at the Sercos S20 bus coupler (S20-S3-BK+). Csos stands for "CIP Safety on Sercos"

The function block is enabled at a rising edge at the "Execute" input and returns with TRUE to the "Done" output after successful execution.

The function block can then be enabled again at a rising edge at the "Execute" input. The user specifies how often the function block is enabled.

The "Active" output displays if the function block is in operation, e.g. calling diagnostics.

The "Done" output is set when either the diagnostics is acknowledged or if it does not have to be acknowledged or it is a standard diagnostics. The "Done" outputs is also set if there was no diagnostics.



The function block H_SIIIAck_SSDx_S20 requires the **Sercos S20 bus coupler S20-S3-BK+** (R911173318).

Target system	Library/folder
MLC	RIH_Safety_SercosIII
MTX	RIH_Safety_SercosIII

Tab. 18-1: Reference table

Interface

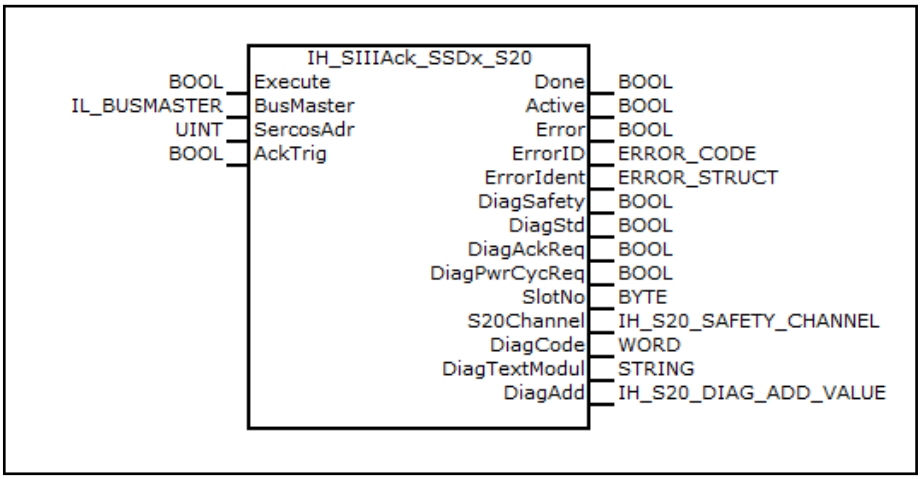


Fig. 18-1: Interface description of the IH_SIIIAck_SSDx_S20 function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Starts the function at a rising edge
	BusMaster	IL_BUSMASTER	Master ID. If multiple masters are simultaneously operated on the control, they are distinguished by the “BusMaster” input variable. If only one master is operated, it has the master ID ="IL_BUSMASTER_0". Also refer to IL_BUSMASTER on page 415
	SercosAdr	UINT	Sercos III address of the S20-S3-BK+ slave
	AckTrig	BOOL	Acknowledgement input: Rising edge required if "DiagAckReq" is present
VAR_OUTPUT	Done	BOOL	TRUE indicates that either the diagnostics is acknowledged or that it odes not have to be acknowledged or that it is standard diagnostics or that there was no diagnostics
	Active	BOOL	TRUE indicates that the function block is active
	Error	BOOL	TRUE indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information See also Troubleshooting, page 724
	DiagSafety	BOOL	TRUE indicates a Safety module diagnostics (Safety diagnostics)
	DiagStd	BOOL	TRUE indicates a standard diagnostics

	Name	Type	Description
	DiagAckReq	BOOL	TRUE indicates that there is a diagnostics that has to be acknowledged and confirmed via "AckTrig".
	DiagPwrCycReq	BOOL	TRUE indicates that a restart/power cycle (switch power off/on) is required to ensure that the S20-SSDx modules run with the "Power On" self test. **Note:** Perform the PowerCycle at the bus coupler. Additional voltage infeed terminals might be required
	SlotNo	BYTE	Slot number of the diagnostic-reporting module
	S20Channel	IH_S20_SAFETY_CHANNEL	Channel for which the diagnostics is pending. Value 255 signifies that the entire module is affected. **Note:** SafetyIoNo, SafetyChannelNo, SafetyIoChannelName are only valid, if "DiagSafety" is TRUE.
	DiagCode	WORD	S20 diagnostic code
	DiagTextModul	STRING	The diagnostic text is read out via the S-0-0095 parameter at S20-S3-BK+. English is default language
	DiagAdd	IH_S20_DIAG_ADD_VALUE	S20 additional diagnostics (code and text)

Tab. 18-2: Inputs/outputs of the IH_SIIIAck_SSDx_S20 function block

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
BusMaster	IL_BUSMASTER	IL_BUSMASTER_0	IL_BUSMASTER_0	IL_BUSMASTER_0	Rising edge at "Execute"
SercosAdr	UINT	1	511	0	Rising edge at "Execute"
AckTrig	BOOL			FALSE	Continuous if "DiagAckReq" is reported

Tab. 18-3: Min./max. and default values of the function block inputs IH_SIIIAck_SSDx_S20

Types and constants

The types and constants facilitate detailed diagnostic information and provide diagnostic texts to the user.

The "IH_S20_DIAG_ADD_VALUE" structure contains the additional diagnostic code and the selected texts.

Name	Type	Comment
Value	DWORD	S20 Additional diagnostic value
Text	STRING	Text for the additional diagnostic value

Tab. 18-4: "IH_S20_DIAG_ADD_VALUE"

The "IH_S20_SAFETY_CHANNEL" structure contains the channel and the input assignment.

Name	Type	Value range
S20ChannelNo	BYTE	0 - 7, 255 (255: whole module is affected)
SafetyIoNo	BYTE	0 - 3, 255 (255: whole module is affected)
SafetyChannelNo	BYTE	1, 2, 255 (255: whole module is affected)
SafetyIoChannelName	STRING	'OUTx_CHy', 'INx_CHy', ''

Tab. 18-5: "IH_S20_SAFETY_CHANNEL"

The "IH_SAFETY_IO_DIAG_TEXT" structure contains diagnostic text and diagnostic code.

Name	Type	Comment
DiagCode	WORD	Diagnostic code
Text	STRING(255)	Diagnostic text

Tab. 18-6: "IH_SAFETY_IO_DIAG_TEXT"

The "SafetyDiagTexts" constant contains two arrays of type "IH_SAFETY_IO_DIAG_TEXT" with diagnostic texts for the S20-SSDx module:

- German: arS20S3SafetyDiagTexts_De
- English: arS20S3SafetyDiagTexts_En

The "IH_SIII_S20_SAFETY_DIAG_TEXT_AMOUNT" constant indicates the number of diagnostic texts.

Search in "arS20S3SafetyDiagTexts_xx" for the "DiagCode" to determine the corresponding diagnostic text.

```
// Get DE diagnosis text from global text array
 IF FB_SIIIAck_SSDx_S20.DiagCode <> 0 THEN
     FOR i:=1 TO IH_SIII_S20_SAFETY_DIAG_TEXT_AMOUNT DO
         IF arS20S3SafetyDiagTexts_De[i].DiagCode = FB_SIIIAck_SSDx_S20.DiagCode THEN
             szStringS20 := concat (szStringS20 , arS20S3SafetyDiagTexts_De[i].Text);
             bTextFoundS20 := TRUE;
             EXIT;  // Text found
         END_IF
     END_FOR
 END_IF
 IF bTextFoundS20 = FALSE THEN
     // If no text found use text from modul
     szStringS20 := concat (szStringS20 , FB_SIIIAck_SSDx_S20.DiagTextModul);
 END_IF
```

Fig. 18-2: Example to determine the diagnostic texts from "arS20S3SafetyDiagTexts_xx"

Functional description

The "IH_SIIIAck_SSDx_S20" function block is used to read out and acknowledge diagnostics of the S20-SSDx modules (Csos modules) at the Sercos S20 bus coupler (S20-S3-BK+).

The "IH_SIIIAck_SSDx_S20" function block is based on the basic function block "IL_ExecuteDoneBase".

There are different function blocks in the function block, such as: IL_SIIIRemoteSlaveState, IL_SIIIRemoteSlaveStateDetails, IL_SIIISvcRead, IL_SIIISvcWrite.

The "IH_SIIIAck_SSDx_S20" function block has to be used by specifying the Sercos address and the Sercos master of the Sercos S20 bus coupler. The

function block contains all S20-SSDx modules located at the Sercos S20 bus coupler.

☞ To keep the communication load on Sercos low, use as few "IH_SIIIAck_PSDx_S20" function blocks as possible simultaneously.

If errors that have to be acknowledged are reported in multiple channels and multiple modules, they are handled subsequently in the function block. Diagnostics have to be acknowledged individually. This means, the diagnostic code is output and a separate edge is required at the "AckTrig" input. TRUE at the "Done" output and FALSE at the "DiagAckReq" and "DiagStd" outputs indicates that all diagnostics were acknowledged.

Acknowledging diagnostics is supported for the following Csos S20-SSDx modules:

- S20-SSDI-8/4
- S20-SSDO-8/3

In case of Safety diagnostics subject to acknowledgements, these are mostly channel-specific diagnostics, such as "Symmetry violation at input x channel y". After troubleshooting, these diagnostics have to be acknowledged before the channel/module can be operated again.

⚠ WARNING **Refer to the S20-SSDx module documentation**

For the meaning of device-specific diagnostics and error codes of the Sercos S20 bus coupler (S20-S3-BK+) and the Csos S20 modules, refer to the respective device documentation. Notes on troubleshooting as well as instructions on error elimination and acknowledgement are described in the respective documentation.

⚠ WARNING **Acknowledgement can cause a dangerous state**

Apart from the specified exceptions, acknowledging an error causes an immediate return of the safe output/input to the operating state. Thus, ensure that the acknowledgement does not cause a dangerous machine state before acknowledging an error! When planning the machine/system, take into account that acknowledging may only be possible if the dangerous area is visible.

Acknowledging and reading out Safety-relevant diagnostics is initiated at a rising edge at the "Execute" input.

Initially, the command configuration at S20-S3-BK+ is read out once. Thus, it is known which slots contain the S20-SSDx modules. If the input parameters "BusMaster" or "SercosAdr" change during the function block runtime, the command configuration at the S20-S3-BK+ is read out again.

First, "IL_SIIIRemoteSlaveState" is executed to detect, whether there is a diagnostics at the Sercos S20 bus coupler or at it modules.

If a diagnostics is pending, the function block checks whether the diagnostics originates from an S20-SSDx module or whether it has to be acknowledged. If yes, acknowledging is prepared.

If the Safety diagnostics has been acknowledged successfully, the next diagnostics pending at the Sercos S20 bus coupler.

Thus, the outputs "DiagSafety", "DiagStd", "DiagAckReq", "DiagPwrCycReq", "SlotNo", "S20Channel" und "DiagCode" are set on this basis.

The "DiagAckReq" output indicates whether a Safety diagnostics subject to acknowledgement is present and whether this diagnostics has to be acknowledged at a positive edge at the "AckTrig" after error elimination. The "ResetDiag object 0x0019" is internally used for acknowledging.

Diagnostics

The "DiagCode" output returns the state of the Sercos S20 bus coupler and the S20 modules.

The "SlotNo" output indicates the diagnostic-reporting slot or the S20 module number.

The "S20Channel" output shows channel reporting the diagnostics. Value 255 signifies that the entire module is affected. The basis is the diagnostic object "DiagState object 0x0018, index 3: Channel/Group/Modules".

The "DiagTextModul" output returns the diagnostic text directly from the module (parameter S-0-0095).

The "DiagAdd" output returns additional diagnostics (code and text). The diagnostic object "DiagStateAddValue object 0x0034" is read out.

Troubleshooting

The " IH_SIIIAck_SSDx_S20" function block forwards the error codes of the internally used function blocks at the individual error output. See table **SERCOS_TABLE(16#0010)**, F_RELATED_TABLE (16#0170).

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#00008097	0	Sercos: Device not available

Tab. 18-7: Example of error codes of the SERCOS_TABLE of the internally used function blocks

The "IH_SIIIAck_SSDx_S20" function block generates the following error codes:

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#000000FB	0	The module ID differs from the module ID configured in the IndraWorks project
DEVICE_ERROR	16#000000FC	0	The number of modules differs from the modules configured in the IndraWorks project
DEVICE_ERROR	16#000000FD	0	The maximum number of 20 subsequent acknowledgements for the same slot is reached.
DEVICE_ERROR	16#000000FE	0	Device not supported. Use bus coupler S20-S3-BK+
STATE_MACHINE_ERROR	16#000000FF	0	Error in state machine. If this error is output, the function block only generated this error.

Tab. 18-8: Error codes generated by IH_SIIIAck_SSDx_S20

19 RIH_Dali.library

19.1 RIH_Dali, general information

DALI - Digital Addressable Lighting Interface is a protocol for lighting and building automation according to the IEC 62386

The **RIH_DALI** library is to be used with the Inline modules **R-IB IL DALI/PWR-PAC** (R911172212) and **R-IB IL DALI-PAC** (R911172213). The Inline module can be connected to the control or to an Inline bus coupler (Profibus, Profinet, Sercos).

With one DALI control (**R-IB IL DALI/PWR-PAC**) up to 64 devices can be accessed (e.g. lamps, electrical ballasts, sensors or dimmers). DALI modules (**R-IB IL DALI-PAC**) can be connected to the DALI control to extend the DALI network. Bosch Rexroth provides lamps with DALI support like SL36 or SL78 electronic. The DALI network can be created in line or star topology. Each device can be addressed individually or by broadcast commands. DALI ballasts can be arranged and accessed in groups (max. 16 groups) or scenes (max. 16 scenes).

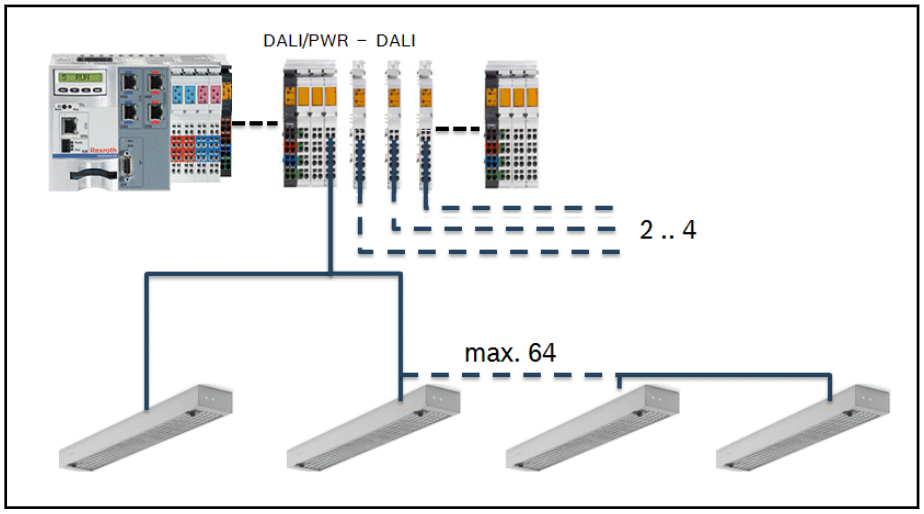


Fig. 19-1: DALI network example with Lx5 control, R-IB IL DALI/PWR-PAC and SL36 electronic

The RIH_DALI library provides a basic (all to start with DALI) and an advanced (additional support) part. With the Bosch Rexroth DALI function block solution, it is possible to control a DALI network without the need for any other tool (no external tool is needed to set up the DALI addresses initially, can be done using the function block **IH_DaliProgramShortAddress**).

To start a PLC program with RIH_DALI, the function block **IH_DaliReadWrite** has to be instantiated for each DALI Inline module and provides the communication handle for each other function block in the library RIH_DALI, e.g. IH_DALIOn, IH_DALIDim.

Function block of the RIH_Dali library

Type	Description
Basic function blocks (Basic)	
IH_DaliReadWrite, page 742	Master function block that ensures the communication with the DALI Inline module and provides the communication handle **IOstrToken** for the other function blocks

Type	Description
IH_DaliDim, page 726	Controls the brightness of the ballast, can be used for groups or broadcast (switch on the ballast if required), the brightness level is defined
IH_DaliOff, page 751	Switch off the ballast, can be used for groups or broadcast
IH_DaliOn, page 753	Switch on the ballast, can be used for groups or broadcast
IH_DaliDirectPower, page 734	Controls the brightness of the ballast, can be used for groups or broadcast (switch the ballast on, if necessary), the brightness can be specified
IH_DaliAddress, page 728	Assigns a new address to the ballast
IH_DaliCallScene, page 730	Changes the brightness of the scene to the defined value
IH_DaliAddToGroup, page 747	Adds the ballast to a group
IH_DaliAddToScene, page 749	Adds the ballast or a group to a scene
IH_DaliRemoveFromGroup, page 732	Removes the ballast of the group
IH_DaliRemoveFromScene, page 736	Removes the ballast or a group from the scene
Advanced function blocks	
IH_DaliQueryStatus, page 738	Reads status of all connected DALI ballasts
IH_DaliQueryActualLevel, page 744	Reads the actual power of the ballast
IH_DaliQuerySceneLevel, page 757	Reads the actual power of the scene
IH_DaliQueryDTR, page 769	Reads the Data Transfer Register (DTR) of the ballast
IH_DaliStoreParameter, page 763	Stores the content of the Data Transfer Register (DTR) to a specified parameter of the ballast
IH_DaliQueryParameter, page 771	Reads the parameters Fade Rate, Fade Time, Min Level, Max Level, Power On Level, Physical Min Level and System Failure Level of the ballast
IH_DaliQueryGroup, page 774	Reads the assigned group of the ballast
IH_DaliQueryRandomAddress, page 777	Reads the random address of the ballast
IH_DaliSearchAddress, page 755	Stores the search address of a ballast
IH_DaliProgramShortAddress, page 759	Assigns a short address to a ballast by random address allocation or physical selection during initialization
IH_DaliReset, page 766	Reset variables in the persistent memory of the ballast

Tab. 19-1: Function block of the RIH_Dali library

19.2 IH_DaliDim

Brief description The function block **IH_DaliDim** is used to control the brightness level of the ballast. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-2: Reference table

Interface description

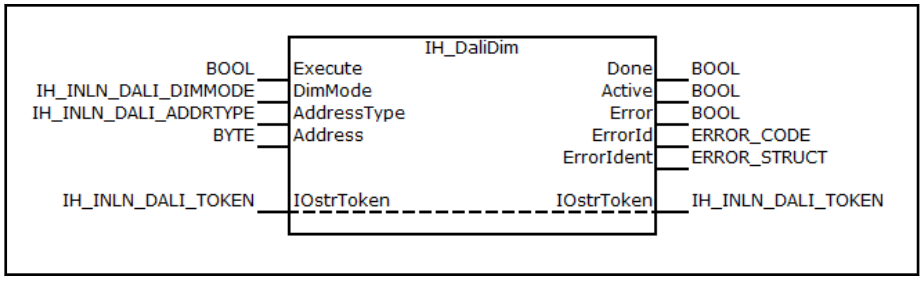


Fig. 19-2: IH_DaliDim function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	DimMode	IH_INLN_DALI_DIMMODE	Enumeration to select the dim mode. IH_INLN_DALI_DIMMODE, page 781
	AddressType	IH_INLN_DALI_ADDRTYPE	Enumeration to select type of the address. IH_INLN_DALI_ADDRTYPE, page 781
	Address	BYTE	Address depends on the type of selection made in the AddressType input. The value ranges from 0 to 63 for short addresses, 0 to 15 for group addresses.
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Class of error
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks. IH_INLN_DALI_TOKEN, page 780

Tab. 19-3: Interface variables of the IH_DaliDim function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
DimMode	IH_INLN_DALI_DIMMODE	UP	STEP-DOWN_AND_OFF	UP	Rising edge at Execute.
AddressType	IH_INLN_DALI_ADDRTYPE	SHORTADDR	BROADCAST	SHORTADDR	Rising edge at Execute.
Address	BYTE	0	Short Address = 63 Group address = 15	0	Rising edge at Execute.

Tab. 19-4: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliDim

Functional description

This function block is used to increase or decrease the brightness level of individual ballast or group or all connected ballasts as per the selection made in the Address type input. The user can select the different dimming modes in the DimMode input. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active", the output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliDim function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between IH_DaliReadWrite and IH_daliDim function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliDim function block is configured in ON_AND_STEPUP mode for single ballast. In this mode if the ballast is in off state, this command initially enables the ballast and sets it to min level. A new rising edge at the bExecute input increases the light level by one step.

Variable declaration

```
VAR
    fbIH_DaliDim: IH_DaliDim;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Example

```
fbIH_DaliDim(
    Execute:= bExecute,
    DimMode:= IH_INLN_DALI_DIMMODE.ON_AND_STEPUP,
    AddressType:= IH_INLN_DALI_ADDRTYPE.SHORTADDR,
    Address:= BYTE#2,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.3 IH_DaliAddress

Brief description

The function block **IH_DaliAddress** assigns a new short address to ballast. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-5: Reference table

Interface description

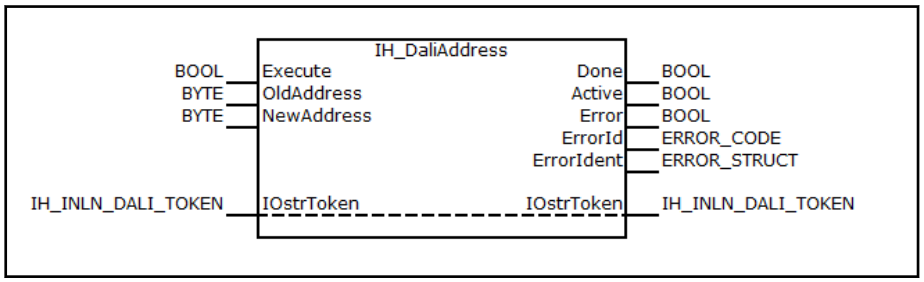


Fig. 19-3: IH_DaliAddress function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	OldAddress	BYTE	Old address of ballast (0 to 63)
	NewAddress	BYTE	New address of ballast (0 to 63)
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks. IH_INLN_DALI_TOKEN, page 780

Tab. 19-6: Interface variables of the IH_DaliAddress function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
OldAddress	BYTE	0	63	0	Rising edge at Execute
NewAddress	BYTE	0	63	0	Rising edge at execute

Tab. 19-7: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliAddress

Functional description

The function block assigns a new short address to ballast. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs

during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information.

The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliAddress function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between IH_DaliReadWrite and IH_DaliAddress function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliAddress function block is configured to change the address of ballast from 2 to 4. A new rising edge at the bExecute input assigns the new address to ballast.

Variable declaration

```
VAR
    fbDaliAddress : IH_DaliAddress;
    bExecute      : BOOL;
    stDaliToken   : IH_INLN_DALI_TOKEN;
    bDone         : BOOL;
    bActive       : BOOL;
    bError        : BOOL;
    enErrorID     : ERROR_CODE;
    stErrorIdent  : ERROR_STRUCT;
END_VAR
```

Example

```
fbDaliAddress(
    Execute:= bExecute,
    OldAddress:= BYTE#2,
    NewAddress:= BYTE#4,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.4 IH_DaliCallScene

Brief description

The function block **IH_DaliCallScene** is used to call the predefined scene. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library / Folder
MLC	RIH_Dali / Basic
MTX	RIH_Dali / Basic
MLD (IL 2G)	RIH_Dali / Basic

Tab. 19-8: Reference table

Interface description

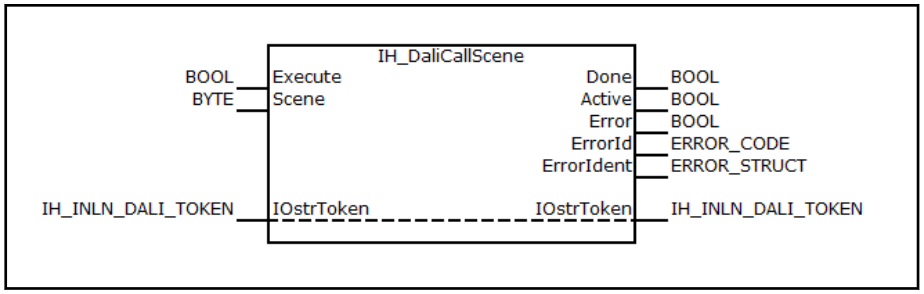


Fig. 19-4: IH_DaliCallScene function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	Scene	BYTE	Scene number Value ranges from 0 to 15
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks. IH_INLN_DALI_TOKEN, page 780

Tab. 19-9: Interface variables of the IH_DaliCallScene function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Scene	BYTE	0	15	0	Rising edge at Execute

Tab. 19-10: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliCallScene

Functional description

This function block calls the predefined scene. A total of 16 scenes can be called. The scene value ranges from 0 to 15. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. The modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. The ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliCallScene function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between IH_DaliReadWrite and IH_DaliCallScene function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, IH_DaliCallScene is used to call the predefined scene number 5. A rising edge at the execute input activates scene 5 on all defined ballasts.

Variable declaration

```
VAR
    fbIH_DaliCallScene: IH_DaliCallScene;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Example

```
fbIH_DaliCallScene(
    Execute:= bExecute,
    Scene:= BYTE#5,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.5 IH_DaliRemoveFromGroup

Brief description

The function block **IH_DaliRemoveFromGroup** removes ballast from the group specified. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-11: Reference table

Interface description

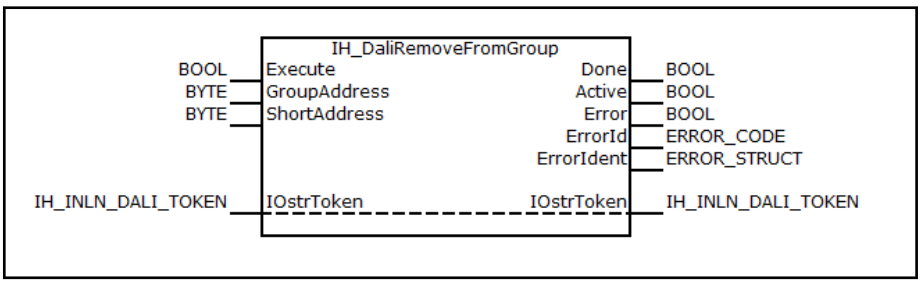


Fig. 19-5: IH_DaliRemoveFromGroup function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	GroupAddress	BYTE	Address of the group (0 to 15)
	ShortAddress	BYTE	Address of the ballast (0 to 63)
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-12: Interface variables of the IH_DaliRemoveFromGroup function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
GroupAddress	BYTE	0	15	0	Rising edge at Execute
ShortAddress	BYTE	0	63	0	Rising edge at Execute

Tab. 19-13: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliRemoveFromGroup

Functional description

The function block removes ballast from the specified group. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during operation the error bit is set to TRUE, ErrorID and ErrorIdent displays the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliRemoveFromGroup function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliRemoveFromGroup function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliRemoveFromGroup function block is configured to remove ballast (Address = 2) from the specified group number 5. A new rising edge at bExecute input removes the ballast from the assigned group.

Variable declaration

Program:

```
VAR
    fbDaliRemoveFromGroup: IH_DaliRemoveFromGroup;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Example

```
fbDaliRemoveFromGroup(
    Execute:= bExecute,
    GroupAddress:= BYTE#5,
    ShortAddress:= BYTE#2,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.6 IH_DaliDirectPower

Brief description

The function block **IH_DaliDirectPower** is used to provide direct input brightness level. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-14: Reference table

Interface description

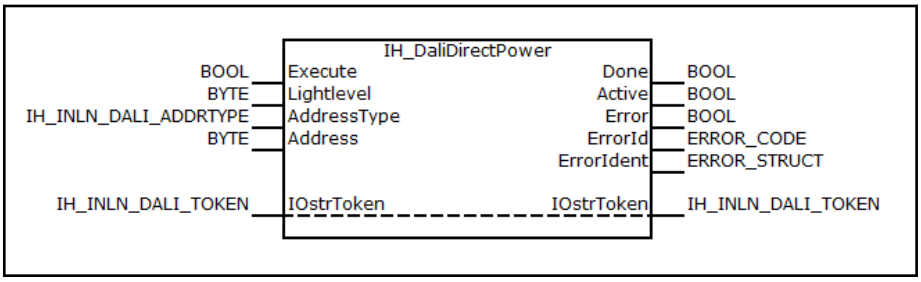


Fig. 19-6: IH_DaliDirectPower function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	LightLevel	BYTE	The value indicates the brightness level. The value ranges from 0 to 254
	AddressType	IH_INLN_DALI_ADDR-TYPE	Enumeration to select type of the address. IH_INLN_DALI_ADDRTYPE, page 781
	Address	BYTE	Address depends on the type of selection made in the AddressType input. The value ranges from 0 to 63 for short addresses, and from 0 to 15 for group addresses
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TO-KEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-15: Interface variables of the IH_DaliDirectPower function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
LightLevel	BYTE	0	254	254	Rising edge at Execute
AddressType	IH_INLN_DA-LI_ADDRTYPE	SHORTADDR	BROADCAST	SHORTADDR	Rising edge at Execute
Address	BYTE	0	Short Address = 63 Group address = 15	0	Rising edge at Execute

Tab. 19-16: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliDirectPower

Functional description

This function block is used to provide the direct input brightness level of individual ballast or a group or all connected ballasts as per the selection made in the Address type input. The power is calculated from the following formula

$$P_x = 10^{\left(\frac{x-1}{253/3}\right)} \cdot \frac{P_{100\%}}{1000}$$

Fig. 19-7: Power calculation

Here, X represents the value of the LightLevel input.

Activate the function block after the first PLC cycle by enabling the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. The ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, sercos III and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliDirectPower function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_daliDirectPower function blocks, so the same variable should be assigned at IOstrToken in both function blocks.

In the example, IH_DaliDirectPower is used to set the light level of single ballast to 254.

Variable declaration

```
VAR
    fbIH_DaliDirectPower: IH_DaliDirectPower;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Example

```
fbIH_DaliDirectPower(
    Execute:= bExecute,
    LightLevel:= BYTE#254,
    AddressType:= IH_INLN_DALI_ADDRTYPE.SHORTADDR,
    Address:= BYTE#2,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.7 IH_DaliRemoveFromScene

Brief description The function block **IH_DaliRemoveFromScene** removes ballast/group from the scene specified. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-17: Reference table

Interface description

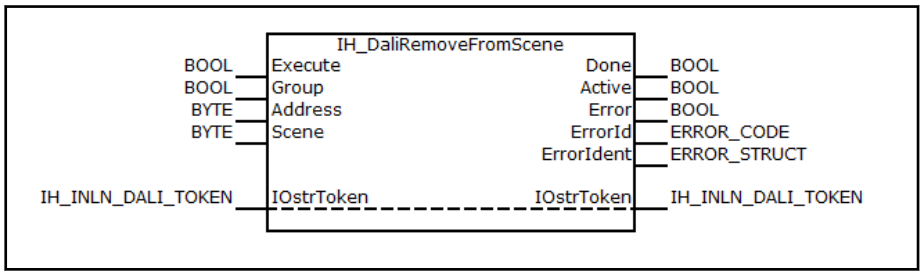


Fig. 19-8: IH_DaliRemoveFromScene function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	Group	BOOL	True= Address is group address FALSE= Address is ballast address
	Address	BYTE	Address of the ballast/group (depends on input group)
	Scene	BYTE	Scene number (0 to 15)
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-18: Interface variables of the IH_DaliRemoveFromScene function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Group	BOOL	FALSE	TRUE	FALSE	Rising edge at Execute
Address	BYTE	0	Group=15 Ballast=63	0	Rising edge at Execute
Scene	BYTE	0	15	0	Rising edge at Execute

Tab. 19-19: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliRemoveFromScene

Functional description

The function block removes the ballast/group from the specified scene.

Activate the function block after the first PLC cycle by enabling the "Execute" input to TRUE. Modified input values are only accepted upon a new Rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized upon a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. The ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliRemoveFromScene function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between IH_DaliReadWrite and IH_DaliRemoveFromScene function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliRemoveFromScene function block is configured to remove ballast (Address = 2) from the specified scene number 5. The new rising edge at the bExecute input removes the ballast from an assigned scene.

Variable declaration

```
VAR
    fbDaliRemoveFromScene: IH_DaliRemoveFromScene;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bGroup: BOOL;
END_VAR
```

Example

```
fbDaliRemoveFromScene(
    Execute:= bExecute,
    Group:= bGroup,
    Address:= BYTE#2,
    Scene:= BYTE#5,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.8 IH_DaliQueryStatus

Brief description The function block **IH_DaliQueryStatus** is used to read the status of ballasts connected to a Dali bus. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-20: Reference table

Interface description

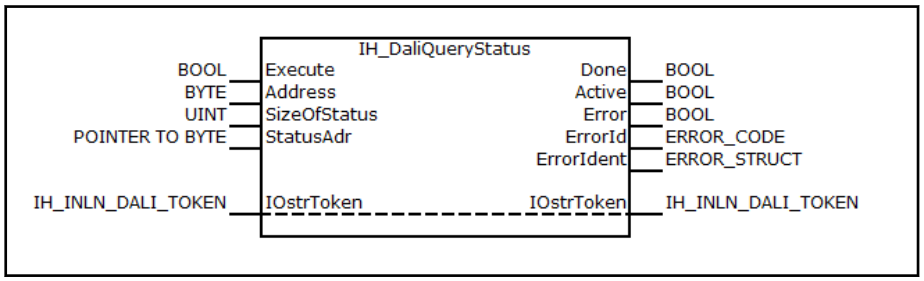


Fig. 19-9: IH_DaliQueryStatus function block

I/O Type	Name	Type	Description		
VAR_INPUT	Execute	BOOL	Rising edge starts calculation		
	Address	BYTE	Address of the ballast (0 to 63 or 255)		
			Address	**SizeOfStatus**	**No of Ballast read**
			0-63	1	1 ballast
			255	64	64 Ballasts
			255	No. of ballasts Between 1 to 64	No of ballasts mentioned in SizeOfStatus
	SizeOfStatus	UINT	Size of the status (1 to 64)		
	StatusAdr	POINTER_TO_BYTE	Address of the Status array		
			Address value	**Size of status array**	
			0 to 63	1 byte	
			255	No. of ballasts (1 to 64)	
VAR_OUTPUT	Done	BOOL	Calculation finished		
	Active	BOOL	Calculation in progress		
	Error	BOOL	Indicates an error		
	ErrorID	ERROR_CODE	Error class		
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783		
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780		

Tab. 19-21: Interface variables of the IH_DaliQueryStatus function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Address	BYTE	0	255	0	Rising edge at Execute.
SizeOfStatus	UINT	1	64	1	Rising edge at Execute.

Tab. 19-22: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliQueryStatus

Functional description

This function block is used to get the status of ballasts connected to Dali bus. If the value at the address input is 0 to 63, the status of the respective ballast is read. If the value at the address is 255, the FB reads the status of the number of ballasts mentioned in the SizeOfStatus input . The status information is available in the status array. Users should ensure that the value defined at the SizeOfStatus input is the same as the size of the array defined in StatusAdr.

The status of the individual ballast is available in one byte and the status information of the byte is as follows

Bit	Status information
0	Status of the ballast. 0= OK 1: NOT OK
1	Lamp failure. 0= OK
2	Lamp power on. 0= OFF
3	Limit value error. 0= the most recently requested lamp power was either between MIN LEVEL and MAX LEVEL or was OFF
4	Fading completed: 0= fading finished 1: fading active
5	Reset status. 0= No
6	Missing short address. 0= No 1= YES
7	Power supply fault. 0= No, a reset or a lamp power control command has been received since the most recent power up

Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. The modified input values are only accepted upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input.

If an error occurs during the operation, the error bit is set to TRUE. The ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliQueryStatus function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_Da-

liReadWrite and IH_DaliQueryStatus function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

Example 1

The IH_DaliQueryStatus function block is configured to get the status of the single ballast (Address = 2) and the status is stored in the "byAddressStatus" byte.

Variable declaration

```
VAR
    fbDaliQueryStatus: IH_DaliQueryStatus;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    byAddressStatus: BYTE;
END_VAR
```

Implementation

```
fbDaliQueryStatus(
    Execute:= bExecute,
    Address:= BYTE#2,
    SizeOfStatus:= SIZEOF(byAddressStatus),
    StatusAdr:= ADR(byAddressStatus),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

Example 2

The IH_DaliQueryStatus function block is configured to get the status of all ballasts (from address 0 to 63) and the status is stored in the "arAddressStatus" array [1..64] of the byte.

Variable declaration

```
VAR
    fbDaliQueryStatus: IH_DaliQueryStatus;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    arAddressStatus: ARRAY[1..64] OF BYTE;
END_VAR
```

Implementation

```
fbDaliQueryStatus(
    Execute:= bExecute,
    Address:= BYTE#16#FF,
    SizeOfStatus:= SIZEOF(arAddressStatus),
    StatusAdr:= ADR(arAddressStatus),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.9 IH_DaliReadWrite

Brief description

The function block **IH_DaliReadWrite** ensures communication with the Dali terminal. It detects the queued commands of other Dali blocks in the program and ensures their execution. For each Dali terminal, the function block has to be instantiated once.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-23: Reference table

Interface description

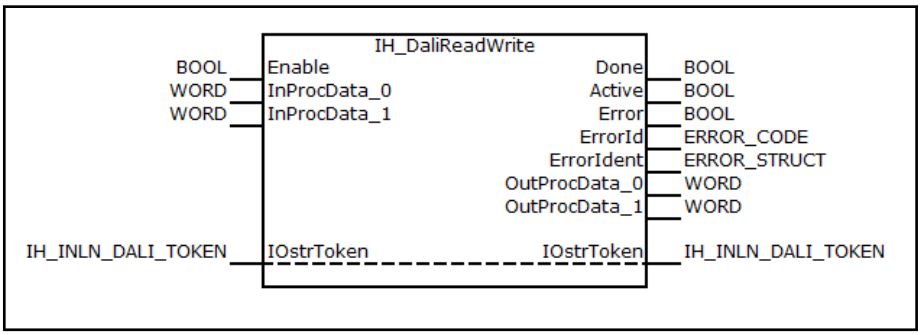


Fig. 19-10: IH_DaliReadWrite function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	InProcData_0	WORD	Input process data word 0
	InProcData_1	WORD	Input process data word 1
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in the next cycle
	Active	BOOL	Function block is working
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
	OutProcData_0	WORD	Output process data word 0
	OutProcData_1	WORD	Output process data word 1
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-24: Interface variables of the IH_DaliReadWrite function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous

Tab. 19-25: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliReadWrite

Functional description

The function block ensures communication with the Dali terminal. The "InProcData" and "OutProcData" should be mapped to Dali terminal. "IostrToken" should link to other Dali blocks in the program. The "IostrToken" structure communicates with the other Dali blocks and exchanges the data. The response command received from the module optionally and forwarded to the other Dali function blocks. In the respective Dali blocks, commands are evaluated and processed. Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If queued commands are detected, the function block will start processing until the "Enable" bit is TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Handling specials

If the Dali module is used in the Profibus or Profinet field bus, the byte order has to be changed to Byte 0/1/2/3 = Plug 4/3/2/1.

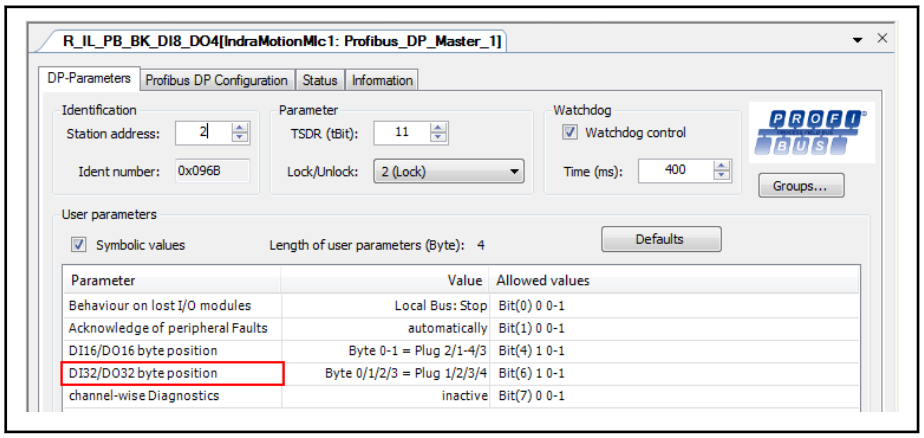


Fig. 19-11: Change of the byte order

Non-functional criteria

Together with the IH_DaliReadWrite function block of the other blocks should be instantiated to communicate with the Dali module.

The current status of the ballast will not be updated automatically. New query required to update the current status.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating other Dali function blocks to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between IH_DaliReadWrite and other Dali function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliReadWrite function block is configured to exchange data with the Dali unit and other Dali function blocks. "wInProcessDa-

ta" and "wOutProcessData" variables should be assigned to the Dali unit for communication.



It might be possible, that some devices may need a DALI address initially, for addressing them. Please use for addressing the function block **IH_DaliProgramShortAddress**

Variable declaration

```
VAR
    fbDaliReadWrite: IH_DaliReadWrite;
    bEnable: BOOL;
    wInProcessData_0: WORD;
    wInProcessData_1: WORD;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    wOutProcessData_0: WORD;
    wOutProcessData_1: WORD;
END_VAR
```

Example

```
fbDaliReadWrite(
    Enable:= bEnable,
    InprocData_0:= wInProcessData_0,
    InprocData_1:= wInProcessData_1,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    OutProcData_0=> wOutProcessData_0,
    OutProcData_1=> wOutProcessData_1);
```

Process Data Mapping

Channels

Variable	Mapping	Channel	Address	Type
		Input0	%IW102	
Application.POU_Dali.wInProcessData_0		Word0	~~%IW102~~	WORD
Application.POU_Dali.wInProcessData_1		Word1	~~%IW104~~	WORD
		Output0	%QW102	
Application.POU_Dali.wOutProcessData_0		Word0	~~%QW102~~	WORD
Application.POU_Dali.wOutProcessData_1		Word1	~~%QW104~~	WORD

Fig. 19-12: Process data mapping IH_DaliReadWrite

19.10 IH_DaliQueryActualLevel

Brief description The function block **IH_DaliQueryActualLevel** is used to read the actual lamp power of ballasts connected to a Dali bus. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library / Folder
MLC	RIH_Dali / Advanced
MTX	RIH_Dali / Advanced
MLD (IL 2G)	RIH_Dali / Advanced

Tab. 19-26: Reference table

Interface description

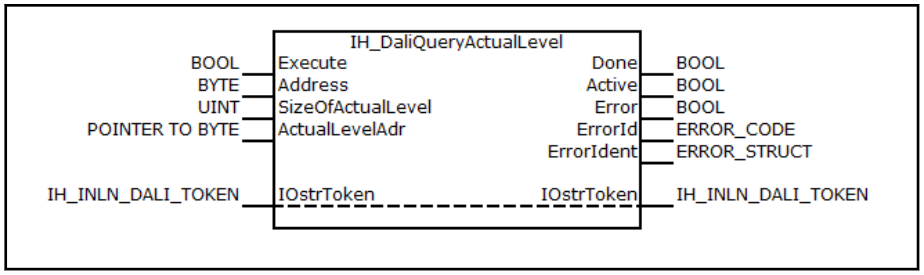


Fig. 19-13: IH_DaliQueryActualLevel function block

I/O type	Name	Data type	Description		
VAR_INPUT	Execute	BOOL	Rising edge starts calculation		
	Address	BYTE	Address of the ballast (0 to 63 or 255)		
			Address	**SizeOfStatus**	**No of Ballast read**
			0-63	1	1 ballast
			255	64	64 ballasts
			255	No. of ballasts. Between 1 to 64.	No of ballasts mentioned in ActualLevel.
	SizeOfActualLevel	UINT	Size of ActualLevel array (1 to 64)		
	ActualLevelAdr	POINTER_TO_BYTE	Address of ActualLevel array		
			Address value	**Size of ActualLevel array**	
			0 to 63	1 byte	
			255	No. of ballasts(1 to 64)	
VAR_OUTPUT	Done	BOOL	Calculation finished		
	Active	BOOL	Calculation in progress		
	Error	BOOL	Indicates an error		
	ErrorID	ERROR_CODE	Error class		
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783		
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780		

Tab. 19-27: Interface variables of the IH_DaliQueryActualLevel function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Address	BYTE	0	255	0	Rising edge at Execute
SizeOfActualLevel	UINT	1	64	1	Rising edge at Execute

Tab. 19-28: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliQueryActualLevel

Functional description

This function block is used to determine the actual lamp power of ballasts connected to the Dali bus. If the value at the address input is 0 to 63, the ActualLevel of the respective ballast is read. If the value at the address is 255, the function block reads the actual lamp power of the number of ballasts mentioned in the SizeOfActualLevel input and the actual lamp power information is available in the ActualLevel array.

Activate the function block after the first PLC cycle by setting the ‘Execute’ input to TRUE. Modified input values are only applied upon a new rising edge at the ‘Execute’ input. During the operation, the "Active" output is set to TRUE. Upon completion, the ‘Done’ bit is set to TRUE. The function block can be reinitialized with a new rising edge at the ‘Execute’ input.

If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the ‘Execute’ input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, sercos III and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliQueryActualLevel function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliQueryActualLevel function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

Example 1

In the example, the IH_DaliQueryActualLevel function block is used to get the actual lamp power of the single ballast.

The function block reads the actual lamp power of ballast 2 and saves it in the byActualLevel variable.

Variable declaration

```
VAR
    fbIH_DaliQueryActualLevel: IH_DaliQueryActualLevel;
    bExecute: BOOL;
    byActualLevel: BYTE;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;

END_VAR
```

Implementation

```
fbIH_DaliQueryActualLevel(
    Execute:= bExecute,
    Address:= BYTE#2,
    SizeOfActualLevel:= SIZEOF(byActualLevel),
    ActualLevelAdr:= ADR(byActualLevel),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

Example 2

In the example, the IH_DaliQueryActualLevel function block is used to get the actual lamp power of all 63 ballasts.

The function block reads the actual lamp power of 63 ballasts and saves it in the arActualLevel array.

An array of 63 bytes should be created and the address of the array should be assigned to ActualLevelAdr.

The size of the array should be mentioned in the SizeOfActualLevel. If the actual array size is less than the value in the SizeOfActualLevel variable, the data can be written to another address and malfunctions can occur in the application.

Variable declaration

```
VAR
    fbIH_DaliQueryActualLevel: IH_DaliQueryActualLevel;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    arActualLevel: ARRAY [0..63] OF BYTE;
END_VAR
```

Implementation

```
fbIH_DaliQueryActualLevel(
    Execute:= bExecute,
    Address:= BYTE#16#FF,
    SizeOfActualLevel:= SIZEOF(arActualLevel),
    ActualLevelAdr:= ADR(arActualLevel),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.11 IH_DaliAddToGroup

Brief description The function block **IH_DaliAddToGroup** inserts ballast in the specified group. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-29: Reference table

Interface description

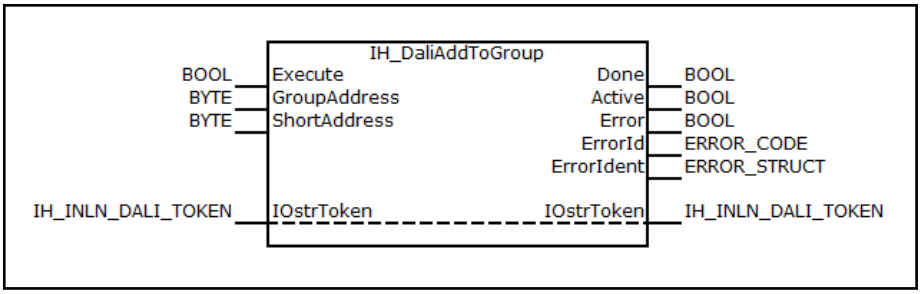


Fig. 19-14: IH_DaliAddToGroup function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	GroupAddress	BYTE	Address of group (0 to 15)
	ShortAddress	BYTE	Address of ballast (0 to 63)
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-30: Interface variables of the IH_DaliAddToGroup function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
GroupAddress	BYTE	0	15	0	Rising edge at Execute
ShortAddress	BYTE	0	63	0	Rising edge at Execute

Tab. 19-31: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliAddToGroup

Functional description

The function block IH_DaliAddToGroup allows the user to insert one or more ballasts into a group. Up to 16 groups can be created. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliAddToGroup function block to communicate with the Dali module. The in/out variable stDaliToken exchanges data between the IH_DaliReadWrite and IH_DaliAddToGroup function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliAddToGroup function block is configured to add ballast with address 2 to group number 10. The new rising edge at the bExecute input adds ballast to the respective group number.

Variable declaration

```
VAR
    fbDaliAddToGroup     : IH_DaliAddToGroup;
    bExecute              : BOOL;
    stDaliToken           : IH_INLN_DALI_TOKEN;
    bDone                 : BOOL;
    bActive               : BOOL;
    bError                : BOOL;
    enErrorID             : ERROR_CODE;
    stErrorIdent          : ERROR_STRUCT;
END_VAR
```

Example

```
fbDaliAddToGroup(
    Execute:= bExecute,
    GroupAddress:= BYTE#10,
    ShortAddress:= BYTE#2,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.12 IH_DaliAddToScene

Brief description

The function block **IH_DaliAddToScene** adds ballast/group to the scene specified. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-32: Reference table

Interface description

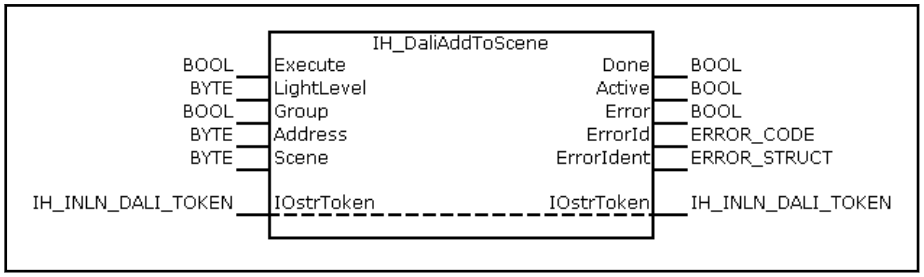


Fig. 19-15: IH_DaliAddToScene function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	LightLevel	BYTE	Intensity of Light (0 to 254)
	Group	BOOL	TRUE= Address is group address FALSE= Address is ballast address
	Address	BYTE	Address of the ballast/group (Depends on input group). Value ranges from 0 to 15 for group and 0 to 63 for ballast
	Scene	BYTE	Scene number (0 to 15)
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-33: Interface variables of the IH_DaliAddToScene function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Light Level	BYTE	0	254	254	Rising edge at Execute
Group	BOOL	FALSE	TRUE	FALSE	Rising edge at Execute
Address	BYTE	0	Group=15 Ballast=63	0	Rising edge at Execute
Scene	BYTE	0	15	0	Rising edge at Execute

Tab. 19-34: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliAddToScene

Functional description

The function block adds ballast or a group to the scene with a specified light level. Participants of same scene can have different light levels. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliAddToScene function block to communicate with the Dali module. The in/out variable stDaliToken exchanges data between the IH_DaliReadWrite and IH_DaliAddToScene function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliAddToScene function block adds ballast or a group of ballasts with address 4 to the scene number 10 with a specified light level as 100. Selection of address type, i.e. short address or goup address depends on the status of the "Group" variable. (TRUE= Address is group address. FALSE= Address is ballast address). The new rising edge at the bExecute input adds ballast or a group to the scene with a specified light level.

Variable declaration

```
VAR
    fbDaliAddToScene     : IH_DaliAddToScene;
    bExecute              : BOOL;
    bGroup            : BOOL;
    stDaliToken           : IH_INLN_DALI_TOKEN;
    bDone                 : BOOL;
    bActive               : BOOL;
    bError                : BOOL;
    enErrorID             : ERROR_CODE;
    stErrorIdent          : ERROR_STRUCT;
END_VAR
```

Example

```
fbDaliAddToScene(
    Execute:= bExecute,
    LightLevel:= BYTE#100,
    Group:= bGroup,
    Address:= BYTE#4,
    Scene:= BYTE#12,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.13 IH_DaliOff

Brief description

The function block **IH_DaliOff** can be used to turn off individual ballast or a group of ballasts or all ballasts. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-35: Reference table

Interface description

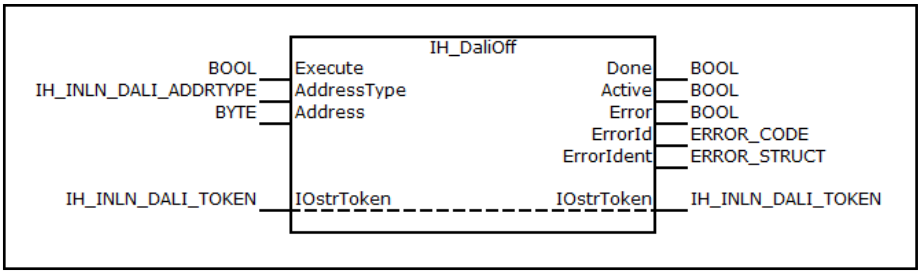


Fig. 19-16: IH_DaliOff function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	AddressType	IH_INLN_DALI_ADDR-TYPE	Enumeration to select type of the address. IH_INLN_DALI_ADDRTYPE, page 781
	Address	BYTE	Address depends on the type of selection Made in AddressType input. The value ranges from 0 to 63 for short address, 0 to 15 for a group address
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TO-KEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-36: Interface variables of the IH_DaliOff function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
AddressType	IH_INLN_DA-LI_ADDRTYPE	SHORTADDR	BROADCAST	SHORTADDR	Rising edge at Execute
Address	BYTE	0	Group=15 Ballast=63	0	Rising edge at Execute

Tab. 19-37: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliOff

Functional description

The function block **IH_DaliOff** can be used to turn off individual ballast or a group of ballasts or all ballasts. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliOff function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliOff function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliOff function block is configured to turn off ballast with address 4 since AddressType is SHORTADDR. A new rising edge at the bExecute input turns off individual ballast or a group of ballasts or all ballasts depending on the Address type selection.

Variable declaration

```
VAR
    fbDaliOff          : IH_DaliOff;
    bExecute               : BOOL;
    stDaliToken            : IH_INLN_DALI_TOKEN;
    bDone                  : BOOL;
    bActive                : BOOL;
    bError                 : BOOL;
    enErrorID              : ERROR_CODE;
    stErrorIdent           : ERROR_STRUCT;

END_VAR
```

Example

```
fbDaliOff(
    Execute:= bExecute,
    AddressType:= IH_INLN_DALI_ADDRTYPE.SHORTADDR,
    Address:= BYTE#4,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.14 IH_DaliOn

Brief description

The function block **IH_DaliOn** can be used to turn on individual ballast or group of ballasts or all ballasts and also sets the brightness level to min, mid or max power level. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Basic
MTX	RIH_Dali/Basic
MLD (IL 2G)	RIH_Dali/Basic

Tab. 19-38: Reference table

Interface description

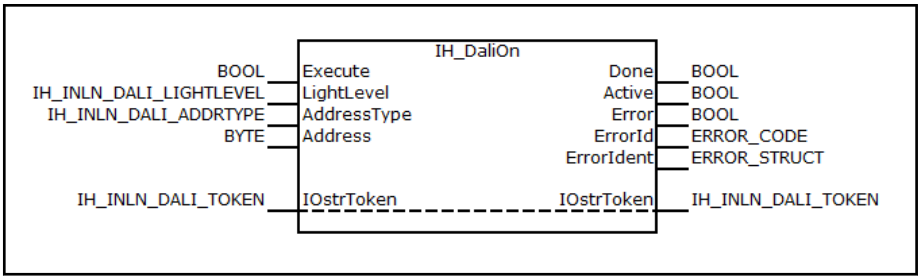


Fig. 19-17: IH_DaliOn function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	LightLevel	IH_INLN_DALI_LIGHT-LEVEL	The value indicates the brightness level. See section structure and enumeration for details IH_INLN_DALI_LIGHTLEVEL, page 781
	AddressType	IH_INLN_DALI_ADDR-TYPE	Enumeration to select type of the address. IH_INLN_DALI_ADDRTYPE, page 781
	Address	BYTE	Address depends on the type of selection made in the AddressType input. The value ranges from 0 to 63 for short address, 0 to 15 for a group address
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TO-KEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-39: Interface variables of the IH_DaliOn function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
LightLevel	IH_INLN_DA-LI_LIGHTLEVEL	MIN_LEVEL	MAX_LEVEL	MAX_LEVEL	Rising edge at Execute
AddressType	IH_INLN_DA-LI_ADDRTYPE	SHORTADDR	BROADCAST	SHORTADDR	Rising edge at Execute
Address	BYTE	0	Group=15 Ballast=63	0	Rising edge at Execute

Tab. 19-40: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliOn

Functional description

The function block can be used to turn on individual ballast or group of ballasts or all ballasts and also sets the brightness level to min, mid or max pow-

er level as per the selection made in the LightLevel input. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion "Done" bit is set to TRUE. The function block can be reinitialized with a new Rising edge at the "Execute" input. If an error occurs during operation the error bit is set to TRUE, ErrorID and ErrorIdent displays the error information. The error can then be acknowledged with a Rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with **IH_DaliReadWrite** Function block to communicate with Dali module.

Compatibility

Compatible with Inline, Profibus and Profinet field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating IH_DaliOn function block to communicate with Dali module. The in/out variable stDaliToken exchanges the data between IH_DaliReadWrite and IH_DaliOn function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example IH_DaliOn function block turns On ballast with Short address 2 and sets its Light Level to max. New rising edge at bExecute turns on individual ballast or Group of ballasts or all ballasts and also sets the brightness level to min, mid or max power level as per selection made in LightLevel input.

Variable declaration

```
VAR
    fbDaliOn          : IH_DaliOn;
    bExecute               : BOOL;
    stDaliToken            : IH_INLN_DALI_TOKEN;
    bDone                  : BOOL;
    bActive                : BOOL;
    bError                 : BOOL;
    enErrorID              : ERROR_CODE;
    stErrorIdent           : ERROR_STRUCT;
END_VAR
```

Example

```
fbDaliOn(
    Execute:= bExecute,
    LightLevel:= IH_INLN_DALI_LIGHTLEVEL.MAX_LEVEL,
    AddressType:= IH_INLN_DALI_ADDRTYPE.SHORTADDR,
    Address:= BYTE#2,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.15 IH_DaliSearchAddress

Brief description The function block **IH_DaliSearchAddress** stores the Search address given by the user. This function block should be called with function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-41: Reference table

Interface description

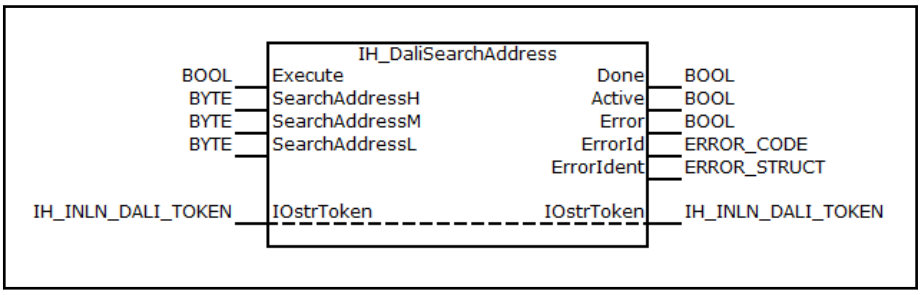


Fig. 19-18: IH_DaliSearchAddress function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	SearchAddressH	BYTE	High byte of Search address
	SearchAddressM	BYTE	Middle byte of Search address
	SearchAddressL	BYTE	Lower byte of Search address
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-42: Interface variables of the IH_DaliSearchAddress function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
SearchAddressH	BYTE	0	255	0	Rising edge at Execute
SearchAddressM	BYTE	0	255	0	Rising edge at Execute
SearchAddressL	BYTE	0	255	0	Rising edge at Execute

Tab. 19-43: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliSearchAddress

Functional description

The function block stores the Search address given by user. Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the

'Execute' input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information.

The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliSearchAddress function block to communicate with the Dali module. The in/out variable stDaliToken exchanges data between the IH_DaliReadWrite and IH_DaliSearchAddress function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliSearchAddress function block is configured to move the value of bySearchAddressH,

bySearchAddressM and bySearchAddressL to the search address of the ballast. The new rising edge at the bExecute input assigns the user input as ballast search address.

Variable declaration

```
VAR
    fbDaliSearchAddress: IH_DaliSearchAddress;
    bExecute: BOOL;
    bySearchAddressH: BYTE;
    bySearchAddressM: BYTE;
    bySearchAddressL: BYTE;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Example

```
fbDaliSearchAddress(
    Execute:= bExecute,
    SearchAddressH:= bySearchAddressH,
    SearchAddressM:= bySearchAddressM,
    SearchAddressL:= bySearchAddressL,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.16 IH_DaliQuerySceneLevel

Brief description The function block IH_DaliQuerySceneLevel is used to read the scene level of the ballast. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-44: Reference table

Interface description

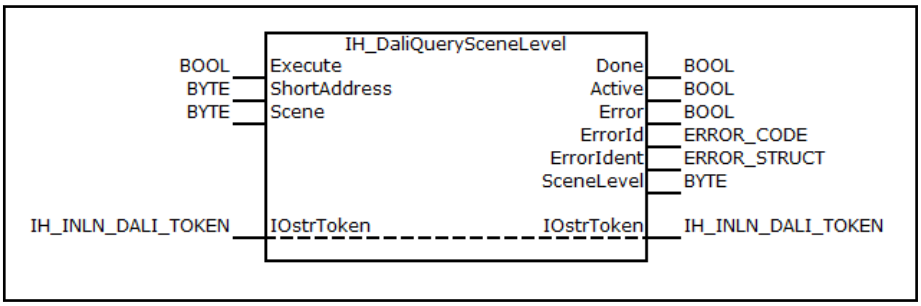


Fig. 19-19: IH_DaliQuerySceneLevel function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	ShortAddress	BYTE	Address of the ballast (0 to 63)
	Scene	BYTE	Scene number. Value ranges from 0 to 15
	Done	BOOL	Calculation finished
VAR_OUTPUT	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
	SceneLevel	BYTE	Scene level of the ballast
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-45: Interface variables of the IH_DaliQuerySceneLevel function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
ShortAddress	BYTE	0	63	0	Rising edge at Execute
Scene	BYTE	0	15	0	Rising edge at Execute.

Tab. 19-46: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliQuerySceneLevel

Functional description

This function block is used to read the scene level of the ballast. The scene value ranges from 0 to 15. If the "SceneLevel" value is 255, the value is invalid or scene is not configured. Activate the function block after the first PLC

cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the IH_DaliReadWrite function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliQuerySceneLevel function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliQuerySceneLevel function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliQuerySceneLevel function block is configured to query the scene level of scene number: 8 from the ballast address: 2. A new rising edge at the bExecute input queries the scene level of a particular ballast.

Variable declaration

```
VAR
    fbDaliQuerySceneLevel: IH_DaliQuerySceneLevel;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bySceneLevel: BYTE;
END_VAR
```

Example

```
fbDaliQuerySceneLevel(
    Execute:= bExecute,
    ShortAddress:= BYTE#2,
    Scene:= BYTE#8,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    SceneLevel=> bySceneLevel);
```

19.17 IH_DaliProgramShortAddress

Brief description

The function block **IH_DaliProgramShortAddress** assigns a short address for ballasts by random address allocation or physical selection during the initialization. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-47: Reference table

Interface description

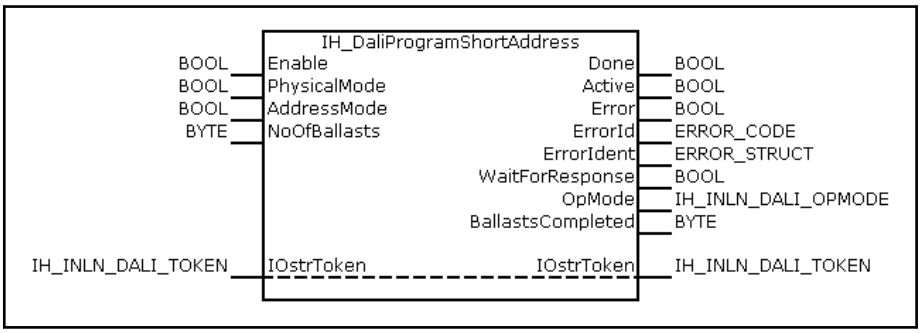


Fig. 19-20: IH_DaliProgramShortAddress function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	PhysicalMode	BOOL	IF TRUE, the physical selection mode is selected. IF FALSE, the random address mode selected
	AddressMode	BOOL	IF TRUE, all ballasts are selected IF FALSE, the ballasts without an address are selected
	NoOfBallasts	BYTE	Number of ballasts to assign to short address
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
	WaitForResponse	BOOL	In PhysicalMode, waiting for physical response from ballasts
	OpMode	IH_INLN_DALI_OPMODE	Current operating mode IH_INLN_DALI_OPMODE, page 782
	BallastsCompleted	BYTE	Total number of ballasts addressing completed
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-48: Interface variables of the IH_DaliProgramShortAddress function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
PhysicalMode	BOOL	FALSE	TRUE	FALSE	Rising edge at Enable
AddressMode	BOOL	FALSE	TRUE	FALSE	Rising edge at Enable
NoOfBallasts	BYTE	1	64	0	Rising edge at Enable

Tab. 19-49: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliProgramShortAddress

Functional description

The function block assigns a short address to ballasts by random address allocation or physical selection during the initialization.

If the PhysicalMode is ‘TRUE’, the ballast is addressed in the physical selection mode, else the random addressing mode is implemented.

Different addressing modes are listed below, to get an overview about the internal behavior of the function block.

Random address mode: (Procedures taking place within the function block during execution)

- Starting the algorithm with command "INITIALIZE" for all ballast or ballast without address
- Send command: "RANDOMIZE": All ballast will choose a binary random number (BRN)
- Finding out search address equal to random address
- The control unit searches the ballast with the lowest BRN by means of the algorithm command "COMPARE". The ballast with the same BRN is found
- The same BRN will be moved to the search address by using IH_DaliSearchAddress.
- The found ballast is assigned a unique 6 bit SHORT ADDRESS (StartAddress) using the command "PROGRAM SHORT ADDRESS"
- Verified by the programmed SHORT ADDRESS with the command "VERIFY SHORT ADDRESS"
- The found ballast shall be removed from the search process by means of command

"WITHDRAW”

- Repeat, starting with step 3 until count reached the "NoOfballasts" value
- If the count reaches the "NoOfBallasts", the process stops with the command "TERMINATE"

Physical address mode: (Procedures taking place within the function block during execution)

- The control unit sends the command "INITIALIZE" for all ballasts or ballasts without address
- The control unit sends command "Physical selection" to enable the physical selection mode of addressing
- The control unit repeats the command QUERY SHORT ADDRESS periodically until a

ballast replies. The ballast responds by removing its lamp

NOTE: The user has to remove and reconnect the lamp of the required ballast

so that it is selected. (Reconnection of lamp should take place within 5 Seconds)

- The selected ballast is assigned a unique 6 bit SHORT ADDRESS using the command "PROGRAM SHORT ADDRESS"
- Finding the Lamp reconnected to ballast with status available using the IH_DaliQueryStatus
- The control unit will send RECALL MIN LEVEL and RECALL MAX LEVEL using the short

address for optical feedback for 5 seconds. This confirms that short address assigned successfully

- Repeat from step 3 until count reached to "NoOfballasts" value
- If the count reaches the "NoOfballasts", the process stops with the command "TERMINATE"

If the AddressMode is "TRUE", all ballast will be selected. If "FALSE", ballasts without short address will be selected for addressing. If the block is waiting for physical response from ballast, the "WaitForResponse" is set to "TRUE". "ballastsCompleted" count the number of completed addressed ballasts. The "OpMode" specified the current operating mode of the function block.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. Modified input values are only applied upon a new rising edge at the "Enable" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information.

The error can then be acknowledged with a rising edge at the "Enable" input.

Handing specials

- During addressing of all ballasts, the lamp should be in ON state
- Once addressing is completed with the IH_DaliProgramShortAddress, to readdress again restart the power supply of the Dali system

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliProgramShortAddress function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the H_DaliReadWrite and IH_ DaliProgramShortAddress function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the examples, the IH_ DaliProgramShortAddress function block is assigning an address to ballasts using the Random addressing mode and the Physical selection mode. A new rising edge at the bExecute input assigns the address to ballasts. Here, the NoOfballasts is assumed as BYTE#20.

Variable declaration

```
VAR
    fbDaliProgramShortAddress: IH_DaliProgramShortAddress;
    bEnable: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bWaitForResponse: BOOL;
    enOpMode: IH_INLN_DALI_OPMODE;
    byBallastsCompleted: BYTE;
END_VAR
```

This function block uses the Random addressing mode for all the available ballasts. The addressing starts from the value zero and increments until no further ballast is found.

Random addressing mode for all ballasts

```
fbDaliProgramShortAddress(
    Enable:= bEnable,
    PhysicalMode:= FALSE,
    AddressMode:= TRUE,
    NoOfBallasts:= BYTE#20,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    WaitForResponse=> bWaitForResponse,
    OpMode=> enOpMode,
    BallastsCompleted=> byBallastsCompleted);
```

This function block uses the Physical selection mode only for ballasts without short address. It will search the available addresses on the network and those addresses will be assigned to the physically selected ballasts.

Physical selection mode for ballasts without Short address

```
fbDaliProgramShortAddress(
    Enable:= bEnable,
    PhysicalMode:= TRUE,
    AddressMode:= FALSE,
    NoOfBallasts:= BYTE#20,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    WaitForResponse=> bWaitForResponse,
    OpMode=> enOpMode,
    BallastsCompleted=> byBallastsCompleted);
```

19.18 IH_DaliStoreParameter

Brief description

The function block **IH_DaliStoreParameter** stores DTR content to the specified ballast parameters such as Fade Rate, Fade Time, Max Level, Min Level, Power On Level, System Failure Level. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-50: Reference table

Interface description

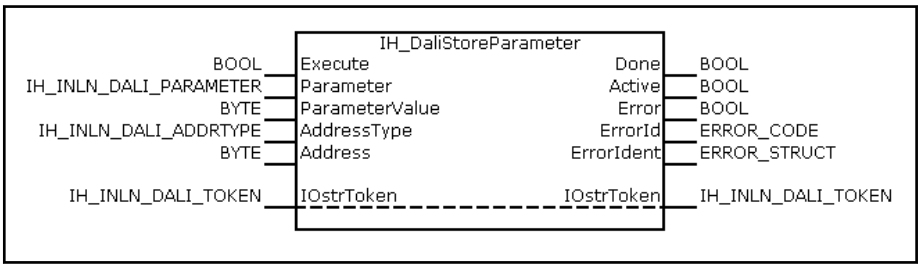


Fig. 19-21: IH_DaliStoreParameter function block

I/O Type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	Parameter	IH_INLN_DALI_PA-RAMETER	Enumeration to select the Parameter. IH_INLN_DALI_PARAMETER, page 783
	ParameterValue	BYTE	Parameter value depends on the selection made in the StoreParameter variable
	AddressType	IH_INLN_DALI_ADDR-TYPE	Enumeration to select the address type IH_INLN_DALI_ADDRTYPE, page 781
	Address	BYTE	Address depends on the type of selection made in the AddressType input. The value ranges from 0 to 63 for short addresses, 0 to 15 for group addresses
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TO-KEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-51: Interface variables of the IH_DaliStoreParameter function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Parameter	IH_INLN_DALI_ PARAMETER	FADE_RATE	SYSTEM_FAIL-URE _LEVEL	FADE_RATE	Rising edge at Execute

Name	Type	Min. value	Max. value	Default value	Effective
ParameterValue	BYTE	-	-	-	Rising edge at Execute
AddressType	IH_INLN_DALI_ADDRTYPE	SHORTADDR	BROADCAST	SHORTADDR	Rising edge at Execute
Address	BYTE	0	Short Address = 63 Group address = 15	0	Rising edge at Execute

Tab. 19-52: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliStoreParameter

Functional description

The function block **IH_DaliStoreParameter** stores DTR content in specified parameters such as Fade Rate, Fade Time, Max Level, Min Level, Power On Level, System Failure Level to individual ballast or a group or all connected ballasts depending on the selection made in the Address type input. Parameters can be selected in the parameter enumeration. Any value for a parameter can be entered using the ParameterValue input variable.

Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliStoreParameter function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliStoreParameter function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the IH_DaliStoreParameter function block stores the value 120 in the parameter Max Level to ballast with short address 1.

Variable declaration

```
VAR
    fbDaliStoreParameter         : IH_DaliStoreParameter;
    bExecute                   : BOOL;
    stDaliToken                : IH_INLN_DALI_TOKEN;
    bDone                      : BOOL;
    bActive                    : BOOL;
    bError                     : BOOL;
    enErrorID                  : ERROR_CODE;
    stErrorIdent               : ERROR_STRUCT;
END_VAR
```

Example

```
fbDaliStoreParameter(
    Execute:= bExecute,
    Parameter := IH_INLN_DALI_PARAMETER.MAX_LEVEL,
    ParameterValue := BYTE#120,
    AddressType:= IH_INLN_DALI_ADDRTYPE.SHORTADDR,
    Address:= BYTE#1,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.19 IH_DaliReset

Brief description

The function block **IH_DaliReset** resets all variables in the persistent memory of the ballast to their reset values. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-53: Reference table

Interface description

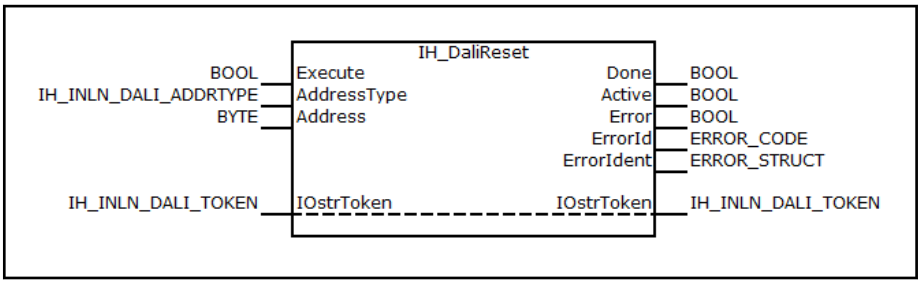


Fig. 19-22: IH_DaliReset function block

I/O Type	Name	Type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	AddressType	IH_INLN_DALI_ADDR-TYPE	Enumeration to select type of the address. IH_INLN_DALI_ADDRTYPE, page 781
	Address	BYTE	Address depends on the type of selection made in the AddressType input. The value ranges from 0 to 63 for short address, 0 to 15 for group address.

I/O Type	Name	Type	Description
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks. IH_INLN_DALI_TOKEN, page 780

Tab. 19-54: Interface variables of the IH_DaliReset function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
AddressType	IH_INLN_DALI_ADDRTYPE	SHORTADDR	BROADCAST	SHORTADDR	Rising edge at Execute.
Address	BYTE	0	Group=15 Ballast=63	0	Rising edge at Execute.

Tab. 19-55: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliReset

Functional description

The function block **IH_DaliReset** resets all variables in the persistent memory of the ballast to their reset values. Standard reset values are mentioned in the table given below. The address type input will select individual ballast or a group or all connected ballasts. If the reset command is in process it is not guaranteed that any other commands are received within the next 300 ms by ballast.

Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.



Reset values shown in the table below may vary depending on the Manufacturer, e.g. Physical Min Level.

Table:

Variable	Reset Value
ACTUAL DIM LEVEL	254
POWER ON LEVEL	254
SYSTEM FAILURE LEVEL	254

Variable	Reset Value
MIN LEVEL	PHYSICAL MIN LEVEL
MAX LEVEL	254
FADE RATE	7 (45 steps/s)
FADE TIME	0 (NO FADE)
SHORT ADDRESS	NO CHANGE
SEARCH ADDRESS	FF FF FF
RANDOM ADDRESS	FF FF FF
GROUP 0-7	0000 0000 (NO GROUP)
GROUP 8-15	0000 0000 (NO GROUP)
SCENE 0-15	255 (MASK)
STATUS INFORMATION	0?100???
	? : Undefined
VERSION NUMBER	NO CHANGE
DEVICE TYPE	0
PHYSICAL MIN LEVEL	FACTORY BURN IN

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of Function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliReset function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliReset function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, IH_DaliReset resets all variables in the persistent memory of the ballast to their reset values. In first example, ballast with Short Address 1 is reset, while in the second example all ballast connected to the Dali bus is reset using the Broadcast command.

Variable declaration

```
VAR
    fbDaliReset             : IH_DaliReset;
    bExecute                  : BOOL;
    stDaliToken               : IH_INLN_DALI_TOKEN;
    bDone                     : BOOL;
    bActive                   : BOOL;
    bError                    : BOOL;
    enErrorID                 : ERROR_CODE;
    stErrorIdent              : ERROR_STRUCT;

END_VAR
```

Program with Addresstype as Short Address

```
fbDaliReset(
    Execute:= bExecute,
    AddressType:= IH_INLN_DALI_ADDRTYPE.SHORTADDR,
    Address:= BYTE#1,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

Program with Addresstype as Broadcast

```
fbDaliReset(
    Execute:= bExecute,
    AddressType:= IH_INLN_DALI_ADDRTYPE.BROADCAST,
    Address:= ,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.20 IH_DaliQueryDTR

Brief description

The function block **IH_DaliQueryDTR** reads contents of the DTR (Data Transfer Register) of ballasts connected to the Dali bus. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library / Folder
MLC	RIH_Dali / Advanced
MTX	RIH_Dali / Advanced
MLD (IL 2G)	RIH_Dali / Advanced

Tab. 19-56: Reference table

Interface description

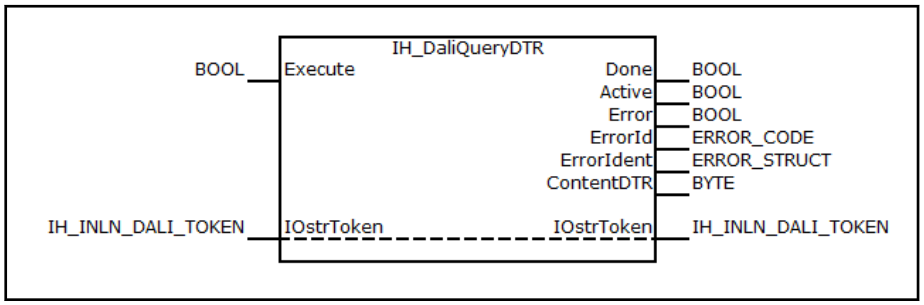


Fig. 19-23: IH_DaliQueryDTR function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
VAR_OUTPUT	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
	ContentDTR	BYTE	DTR content
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-57: Interface variables of the IH_DaliQueryDTR function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous

Tab. 19-58: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliQueryDTR

Functional description

The function block **IH_DaliQueryDTR** reads contents of the DTR (Data Transfer Register) of ballasts connected to the Dali bus. As soon as ‘Execute’ is TRUE, the DTR of all connected ballast will be read and the DTR content value is available in the ‘ContentDTR’ output variable.

Activate the function block after the first PLC cycle by setting the ‘Execute’ input to TRUE. Modified input values are only applied upon a new rising edge at the ‘Execute’ input. During the operation, the „Active" output is set to TRUE. Upon completion, the “Done” bit is set to TRUE. The function block can be reinitialized with a new rising edge at the ‘Execute’ input. If an error occurs during operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the ‘Execute’ input.

Non-functional criteria

This function block should be called with the **IH_DaliReadWrite** function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of Function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliQueryDTR function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliQueryDTR function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

In the example, the function block IH_DaliQueryDTR reads contents of the DTR (Data Transfer Register) of ballasts connected to the Dali bus. As soon

as ‘Execute’ is TRUE, the DTR of all connected ballast will be read and the DTR content value is available in the ‘ContentDTR’ output variable.

Variable declaration

```
VAR

    fbDaliQueryDTR: IH_DaliQueryDTR;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    byContentDTR: BYTE;

END_VAR
```

Example

```
fbDaliQueryDTR(
    Execute:= bExecute,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    ContentDTR=> byContentDTR);
```

19.21 IH_DaliQueryParameter

Brief description

The function block IH_DaliQueryParameter is used to read the parameters fade rate, fade time, min level, max level, power on level, physical min level and system failure level of ballast. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-59: Reference table

Interface description

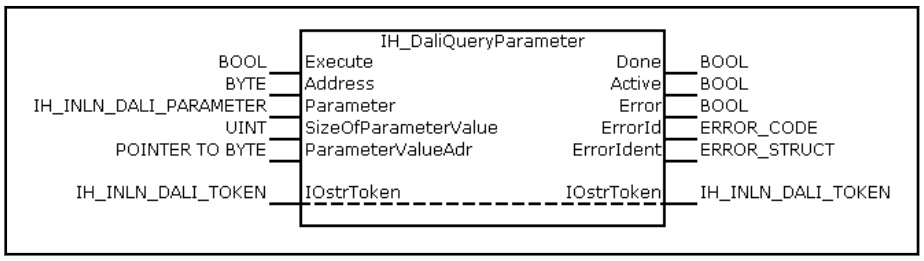


Fig. 19-24: IH_DaliQueryParameter function block

I/O type	Name	Data type	Description		
VAR_INPUT	Execute	BOOL	Rising edge starts calculation		
	Address	BYTE	Address of the ballast (0 to 63 or 255)		
			Address	**SizeOf Parame-terValue**	**No of ballast read**
			0-63	1	1 ballast
			255	64	64 ballasts
			255	No. of ballasts. Between 1 to 64	No of ballasts mentioned in Si-zeOfParameter-Value
	Parameter	IH_INLN_DALI_PA-RAMETER	Enumeration to select the Parameter. See section structure and enumeration for details IH_INLN_DALI_PARAMETER, page 783		
	SizeOfParameterValue	UINT	Size of parameter value (1 to 64)		
	ParameterValueAdr	POINTER_TO_BYTE	Address of ParameterValue. (1 byte or array of 1 to 64)		
			Address value	**Size of ParameterValue array**	
			0 to 63	1 byte	
			255	No. of ballasts (1 to 64)	
VAR_ OUTPUT	Done	BOOL	Calculation finished		
	Active	BOOL	Calculation in progress		
	Error	BOOL	Indicates an error		
	ErrorID	ERROR_CODE	Error class		
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783		
VAR_IN_ OUT	IOstrToken	IH_INLN_DALI_TO-KEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780		

Tab. 19-60: Interface variables of the IH_DaliQueryParameter function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Address	BYTE	0	255	0	Rising edge at Execute
Parameter	IH_INLN_DALI_ PARAMETER	MAX_LEVEL	PHYSI-CAL_MIN_LEVEL	MAX_LEVEL	Rising edge at Execute
SizeOfParameter-Value	UINT	1	64	1	Rising edge at Execute

Tab. 19-61: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliQueryParameter

Functional description

The function block IH_DaliQueryParameter is used to read the ballast parameters. The function block reads the parameter such as fade rate, fade time, min level, max level, power on level, physical min level and system failure level selected from the IH_INLN_DALI_PARAMETER enumeration. The response depends on the selection made in "Address" and "SizeOfParameterValue input".

- If the address is 0-63 and SizeOfParameterValue is 1, the response is 1 byte at ParameterValue
- If the address is 255 and SizeOfParameterValue is 10, the response is 1 byte per ballast (address 0 – 9) , in total 10 bytes at ParameterValue
- If the address is 255 and SizeOfParameterValue is 64, the response is 1 byte per ballast (address 0 – 63, in total 64 bytes at ParameterValue

For FADE_RATE_AND_FADE_TIME, the command response is 1 byte containing XXXX YYYY, where XXXX corresponds to FADE TIME and where YYYY corresponds to FADE RATE. If there is no response from the ballast, the response is assumed as 16#FF at ParameterValue. If the value at the Address input is 0 to 63, the respective ballast is read. If the value at the Address is 255, the function block reads the number of ballasts mentioned in the SizeOfParameterLevel input.

Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the IH_DaliReadWrite function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliQueryParameter function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_DaliQueryParameter function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

Example 1

In the example, the IH_DaliQueryParameter function block is used to get the Max Level of the single ballast.

The function block reads the Max Level power of ballast with short address 2 and saves them in the byParameterValue.variable.

Variable declaration

```
VAR
    fbDaliQueryParameter: IH_DaliQueryParameter;
    bExecute: BOOL;
    byParameterValue: byte;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
```

```
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Implementation

```
fbDaliQueryParameter(
    Execute:= bExecute,
    Address:= BYTE#2,
    Parameter:= IH_INLN_DALI_PARAMETER.MAX_LEVEL,
    SizeOfParameterValue:= SIZEOF(byParameterValue),
    ParameterValueAdr:= ADR(byParameterValue),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

Example 2

In the example, the IH_DaliQueryActualLevel function block is used to get the Min Level power of the all 63 ballasts.

The function block reads the Min Level power of 63 ballasts and stores the arParameterValue array.

The array of 63 bytes should be created and the address of the array should be assigned to ParameterValueAdr.

Make sure that the data array size matches the ballast quantity. If not, serious malfunction can occur in the application in the PLC application.

Variable declaration

```
VAR
    fbDaliQueryParameter: IH_DaliQueryParameter;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    arParameterValue: ARRAY[0..63] OF BYTE;
END_VAR
```

Implementation

```
fbDaliQueryParameter(
    Execute:= bExecute,
    Address:= BYTE#16#FF,
    Parameter:= IH_INLN_DALI_PARAMETER.PHYSICAL_MIN_LEVEL,
    SizeOfParameterValue:= SIZEOF(arParameterValue),
    ParameterValueAdr:= ADR(arParameterValue),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.22 IH_DaliQueryGroup

Brief description The function block **IH_DaliQueryGroup** is used to identify the group of the ballast. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-62: Reference table

Interface description

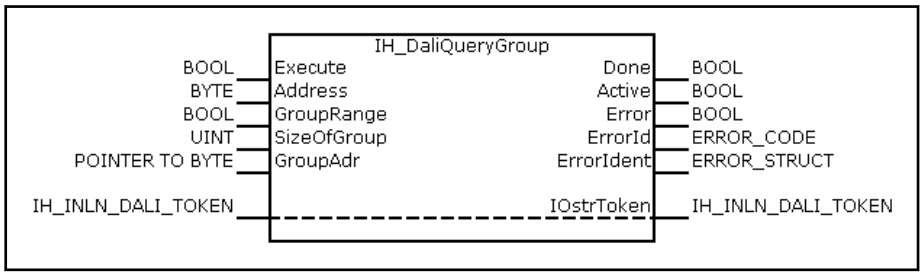


Fig. 19-25: IH_DaliQueryGroup function block

I/O type	Name	Data type	Description		
VAR_INPUT	Execute	BOOL	Rising edge starts calculation		
	Address	BYTE	Address of the ballast (0 to 63 or 255)		
			Address	**SizeOf Group**	**No. of ballast read**
			0-63	1	1 ballast
			255	64	64 ballasts
			255	No. of ballasts. Between 1 to 64	No of ballasts mentioned in SizeOfGroup
	GroupRange	BOOL	FALSE = Query group 0 to 7 TRUE = Query group 8 to 15		
	SizeOfGroup	UINT	Size of Group (1 to 64).		
	GroupAdr	POINTER_TO_BYTE	Address of Group (1 byte or array of 1 to 64)		
			Address value	**Size of Group**	
			0 to 63	1 BYTE	
			255	No. of ballasts (1 to 64)	
VAR_ OUTPUT	Done	BOOL	Calculation finished		
	Active	BOOL	Calculation in progress		
	Error	BOOL	Indicates an error		
	ErrorID	ERROR_CODE	Error class		
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783		
VAR_IN_ OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliReadWrite and other Dali function blocks IH_INLN_DALI_TOKEN, page 780		

Tab. 19-63: Interface variables of the IH_DaliQueryGroup function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Address	BYTE	0	255	0	Rising edge at Execute
GroupRange	BOOL	FALSE	TRUE	FALSE	Rising edge at Execute
SizeOfGroup	UINT	1	64	1	Rising edge at Execute

Tab. 19-64: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliQueryGroup

Functional description

The function block **IH_DaliQueryGroup** is used to identify the group of the ballast.

- If the address is 0-63 and SizeOfGroup is 1, the response is 1 byte at GroupAdr
- If the address is 255 and SizeOfGroup is 10, the response is 1 byte per ballast (address 0 – 9) , in total 10 bytes at GroupAdr
- If the address is 255 and SizeOfGroup is 64, the response is 1 byte per ballast (address 0 – 63), in total 64 bytes at GroupAdr

Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the IH_DaliReadWrite function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliQueryGroup block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_ DaliQueryGroup function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

Example 1

In the example, the IH_ DaliQueryGroup function block is used to get the Max Level of the single ballast.

The function block identifies the group range (0 to 7) of the ballast 2 and stores it in the byQueryGroup variable.

Variable declaration

```
VAR
    fbDaliQueryGroup: IH_DaliQueryGroup;
    bExecute: BOOL;
    byQueryGroup: BYTE;
    stDaliToken: IH_INLN_DALI_TOKEN;
```

```
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
END_VAR
```

Implementation

```
fbDaliQueryGroup(
    Execute:= bExecute,
    Address:= BYTE#2,
    GroupRange:= FALSE,
    SizeOfGroup:=SIZEOF(byQueryGroup),
    GroupAdr:= ADR(byQueryGroup),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

Example 2

In the example, the IH_ DaliQueryGroup function block identifies the group range (8 to 15) of all 63 ballasts.

The function block identifies the group range (8 to 15) of 63 ballasts and stores it in the arQueryGroup array.

An array of 63 bytes should be created and the address of the array should be assigned to GroupAdr.

The size of the array should be specified in the SizeOfGroup. If the actual array size is less than the value mentioned in the SizeOfGroup variable, the data can be written to another address and malfunction can occur in the application.

Variable declaration

```
VAR
    fbDaliQueryGroup: IH_DaliQueryGroup;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    arQueryGroup: ARRAY[0..63] OF BYTE;
END_VAR
```

Implementation

```
fbDaliQueryGroup(
    Execute:= bExecute,
    Address:= BYTE#16#FF,
    GroupRange:= TRUE,
    SizeOfGroup:= SIZEOF(arQueryGroup),
    GroupAdr:= ADR(arQueryGroup),
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent);
```

19.23 IH_DaliQueryRandomAddress

Brief description The function block **IH_DaliQueryRandomAddress** is used to read the random address of ballast. This function block should be called with the function block IH_DaliReadWrite, page 742.

Target system	Library/Folder
MLC	RIH_Dali/Advanced
MTX	RIH_Dali/Advanced
MLD (IL 2G)	RIH_Dali/Advanced

Tab. 19-65: Reference table

Interface description

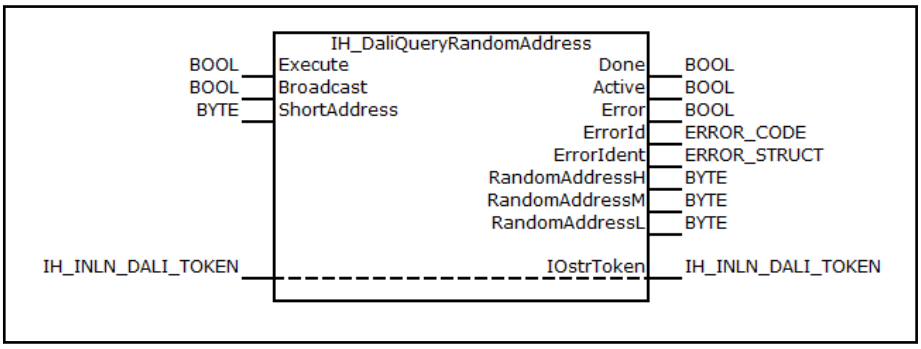


Fig. 19-26: IH_DaliQueryRandomAddress function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts calculation
	Broadcast	BOOL	TRUE = Query random address in broadcast mode. FALSE = Query random address of the ballast specified in ShortAddress
	ShortAddress	BYTE	Short address of the ballast (0 to 63)
	Done	BOOL	Calculation finished
	Active	BOOL	Calculation in progress
VAR_OUTPUT	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Dali, page 783
	RandomAddressH	BYTE	High byte of random address
	RandomAddressM	BYTE	Middle byte of random address
	RandomAddressL	BYTE	Low byte of random address
VAR_IN_OUT	IOstrToken	IH_INLN_DALI_TOKEN	Structure used to handle data between IH_DaliRead-Write and other Dali function blocks IH_INLN_DALI_TOKEN, page 780

Tab. 19-66: Interface variables of the IH_DaliQueryRandomAddress function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Broadcast	BOOL	FALSE	TRUE	FALSE	Rising edge at Execute
ShortAddress	BYTE	0	63	0	Rising edge at Execute

Tab. 19-67: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_DaliQueryRandomAddress

Functional description

The function block **IH_DaliQueryRandomAddress** is used to read the random address of ballast. The function block reads the 8 high bits, 8 mid bits and 8 low bits of a random address. The result is available at the output variables RandomAddressH, RandomAddressM and RandomAddressL.

If the Broadcast input is true, the query of the random address is in broadcast mode. In broadcast mode, single ballast should be connected. Otherwise, while querying, device errors can occur. If the Broadcast input is false, the query of the random address is for the ballast specified in the ShortAddress input.

Activate the function block after the first PLC cycle by setting the "Execute" input to TRUE. Modified input values are only applied upon a new rising edge at the "Execute" input. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. The function block can be reinitialized with a new rising edge at the "Execute" input. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Execute" input.

Non-functional criteria

This function block should be called with the IH_DaliReadWrite function block to communicate with the Dali module.

Compatibility

Compatible with Inline, Profibus, Profinet and sercos III field buses.

Example of a function block

The IH_DaliReadWrite function block should be instantiated before instantiating the IH_DaliQueryRandomAddress function block to communicate with the Dali module. The in/out variable stDaliToken exchanges the data between the IH_DaliReadWrite and IH_ DaliQueryRandomAddress function blocks, so the same variable should be assigned to IOstrToken in both function blocks.

Example 1

In the example, the IH_ DaliQueryRandomAddress function block is used to get the Random address of the single ballast.

The function block reads the Random address of ballast 2 and stores it in the byRandomAddressH, byRandomAddressM and byRandomAddressL variable.

Variable declaration

```
VAR
    fbDaliQueryRandomAddress: IH_DaliQueryRandomAddress;
    bExecute: BOOL;
    stDaliToken: IH_INLN_DALI_TOKEN;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
```

```
    byRandomAddressH: BYTE;
    byRandomAddressM: BYTE;
    byRandomAddressL: BYTE;
END_VAR
```

Implementation

```
fbDaliQueryRandomAddress(
    Execute:= bExecute,
    Broadcast:= FALSE,
    ShortAddress:= BYTE#2,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    RandomAddressH=> byRandomAddressH,
    RandomAddressM=> byRandomAddressM,
    RandomAddressL=> byRandomAddressL);
```

Example 2

The function block reads the Random address of All ballasts and stores it in the byRandomAddressH, byRandomAddressM and byRandomAddressL variable. In Broadcast mode, if any of the ballast's Random address differs from the random address of another ballast, the Dali system throws an Invalid response error.

Implementation

```
fbDaliQueryRandomAddress(
    Execute:= bExecute,
    Broadcast:= TRUE,
    ShortAddress:= ,
    IOstrToken:= stDaliToken,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    RandomAddressH=> byRandomAddressH,
    RandomAddressM=> byRandomAddressM,
    RandomAddressL=> byRandomAddressL);
```

19.24 RIH_Dali library data types

19.24.1 IH_INLN_DALI_TOKEN

This structure is used internally to exchange the command and data between the IH_DaliReadWrite, page 742, and other Dali function blocks.

Library	Folder
RIH_Dali	TYPE

Tab. 19-68: Reference table

Name	Type	Description	Min. value	Max. value	Default value	Application
byRespErr	BYTE	Device error during Communication.	16#0000	16#0F	16#0000	Continuous
byRespSucc	BYTE	Device response during communication	16#0000	16#FF	16#0000	Continuous
byToken	BYTE	Token used by other Dali blocks to communicate with IH_DaliReadWrite FB.	16#0000	16#FF	16#0000	Continuous

Name	Type	Description	Min. value	Max. value	Default value	Application
bRepeat	BOOL	Repeat Dali command again (50ms)	FALSE	TRUE	FALSE	Continuous
wData2	WORD	Command or data processing variable	--------	-------	--------	Continuous

Tab. 19-69: IH_INLN_ DALI_TOKEN

19.24.2 IH_INLN_DALI_ADDRTYPE

This describes the enumeration of the address type.

Library	Folder
RIH_Dali	TYPE

Tab. 19-70: Reference table

ENUM	Description
SHORTADDR	‘Address’ is considered as ballast address
GROUPADDR	‘Address’ is considered as group address
BROADCAST	Broadcast command. Command executes for all ballasts

Tab. 19-71: IH_INLN_DALI_ADDRTYPE

19.24.3 IH_INLN_DALI_DIMMODE

This describes the enumeration for varying light level.

Library	Folder
RIH_Dali	TYPE

Tab. 19-72: Reference table

ENUM	Description
UP	The lamp is made brighter over a period of 200 ms. The rate at which dimming takes place during these 200 ms is given by the FADE RATE variable
STEPUP	Increase the lamp power by one step
ON_AND_STEPUP	Switch the lamp on if necessary, and increase the lamp power by one step.
DOWN	The lamp is made dimmer over a period of 200 ms. The rate at which dimming takes place during these 200 ms is given by the FADE RATE variable
STEPDOWN	Reduce the lamp power by one step
STEPDOWN_AND_OFF	Reduce the lamp power by one step. Switch the lamp off, if actual level is at Min level

Tab. 19-73: IH_INLN_DALI_DIMMODE

19.24.4 IH_INLN_DALI_LIGHTLEVEL

This describes the enumeration of light level.

Library	Folder
RIH_Dali	TYPE

Tab. 19-74: Reference table

ENUM	Description
MIN_LEVEL	Light level is set to minimum level
MID_LEVEL	Light level is set to 50 percent level
MAX_LEVEL	Light level is set to maximum level

Tab. 19-75: IH_INLN_DALI_LIGHTLEVEL

19.24.5 IH_INLN_DALI_OPMODE

Shows the current operating mode of the system.

Library	Folder
RIH_Dali	TYPE

Tab. 19-76: Reference table

Name	Value	Comment
VALUE_1_TO_DTR	0	Pushes the Value 1 to DTR.
VALUE255_TO_DTR	1	Command : For pushing value to Data transfer register.
MASKING_ALL_ADDRESS	2	Masking The Address of all ballasts. (Deleting the address.)
COMMAND_INITIALIZE	3	Command:initialize.
COMMAND_RANDOMISE	4	Command: Randomise.
SET_SEARCHADDRESS	5	Set Search address using IH_DaliSearchAddress.
COMMAND_COMPARE	6	Command: Compare.
DECREMENTING_SEARCH-ADDR	7	Decrementing the Search address for Compare instruction being 'true'.
INCREMENTING_SEARCH-ADDR	8	Incrementing the Search address for Compare instruction being 'false'.
ENABLE_PHYSICAL_SELEC-TION	9	Command: Enable Physical selection.
FIND-ING_AVAILABLE_ADDRESS	10	Finding the next available addresses using the IH_DaliQueryStatus.
QUERY_SHORT_ADDR_IN_PH YSICAL	11	Command: Query Short Address in Physical Selection mode.
PROGRAM_SHORT_ADDR	12	Command: Program Short Address.
VERIFY_SHORT_ADDR	13	Command: Verify Short Address.
VERIFY_LAMP_ON	14	Finding the lamp ON status in selected ballast using the IH_DaliQueryStatus.
RECALL_MINLEVEL	15	Command: Recall MinLevel in address assigned Ballast.
RECALL_MAXLEVEL	16	Command: Recall MaxLevel in address assigned Ballast.
WITHDRAWAL	17	Command: WithDrawal.
TERMINATE	18	Command: Terminate.
COMMAND_INITIALIZE1	19	Command:initialize for first time.
WITHDRAWAL1	20	withdrawal for first time.
TERMINATE1	21	Command: Terminate for first time.

Name	Value	Comment
SEARCH_ADDR_TO_ZERO	22	Command: Search Address to zero.
RESET_ALL_BALLASTS	23	

Tab. 19-77: ENUM IH_INLN_DALI_OPMODE

19.24.6 IH_INLN_DALI_PARAMETER

Enumeration to select the Parameter.

Library	Folder
RIH_Dali	TYPE

Tab. 19-78: Reference table

Name	Value	Comment
FADE_RATE	0	The FADE RATE specifies the rate at which changes are made (in steps per second) in the value of the lamp's power calculated from formula. Allowed limit (1 - 15).
FADE_TIME	1	The FADE TIME specifies the time allowed for the current lamp power to be changed to the requested value calculated from formula. In the case of a lamp that is switched off, the pre-heating and ignition time is not included in the fade time. Allowed limit (0 - 15).
MAX_LEVEL	2	The ballast internally restricts the maximum value of the output power to the lamp by means of the MAX LEVEL variable. Allowed limit (Min Level - 254).
MIN_LEVEL	3	The ballast internally restricts the minimum value of the output power to the lamp by means of the MIN LEVEL variable. Allowed limit (Physical Min Level - Max Level).
POWER_ON_LEVEL	4	When power is supplied to the ballast the lamp is driven to the level of power specified in the variable POWER ON LEVEL.Allowed limit (1 - 254).
SYSTEM_FAILURE_LEVEL	5	If a fault occurs on the DALI bus (the idle voltage remains below the specified level for longer than 500 ms) then the lamp is driven to the power specified by the SYSTEM FAILURE LEVEL variable. Allowed limit (0 - 255).
FADE_RATE_AND_FADE_TIME	6	Query command for FADE RATE and FADE TIME. Answer shall be XXXX YYYY where XXXX corresponds with command FADE RATE and where YYYY corresponds with the command FADE TIME.
PHYSICAL_MIN_LEVEL	7	

Tab. 19-79: ENUM IH_INLN_DALI_PARAMETER

19.25 Error Handling of Library RIH_Dali

The function blocks use error table **INLINEIO_TABLE**, 16#0190.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#3017	16#0000	The ‘OldAddress’ or ‘NewAddress’ input is not valid (<0 or >63)
INPUT_INVALID_ERROR	16#3017	16#0001	The ‘GroupAddress’ is not valid (0< GroupAddress >15)
INPUT_INVALID_ERROR	16#3017	16#0002	The ‘ShortAddress’ input is not valid (0< ShortAddress >63)

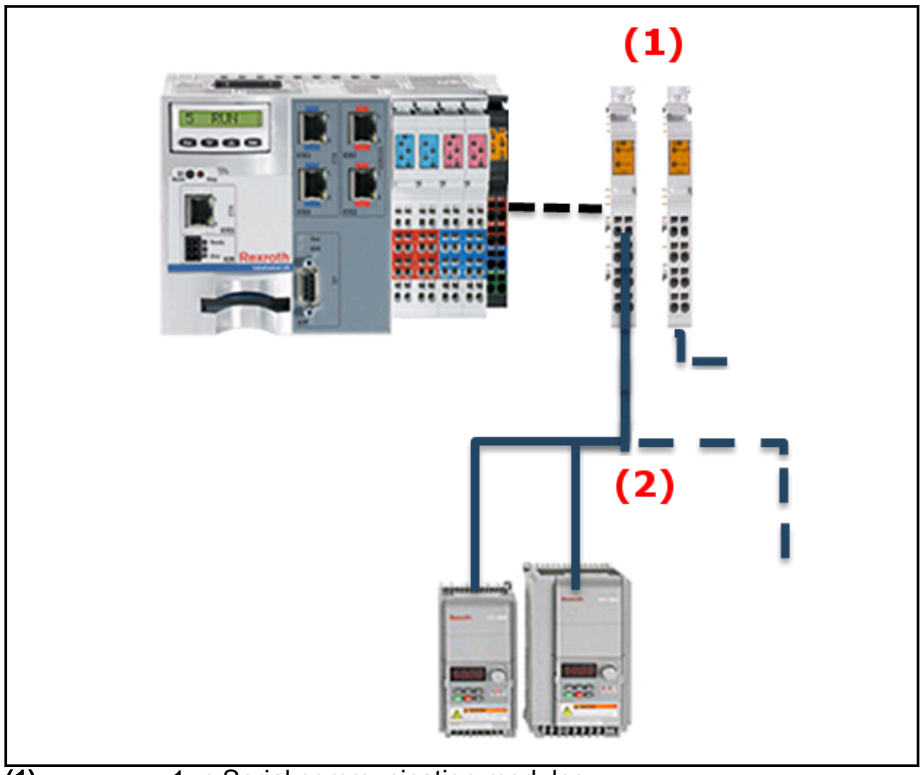
ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#3017	16#0003	The ‘LightLevel’ input is not valid (0 or IH_INLN_DALI_LIGHTLEVEL.MIN_LEVEL < LightLevel >254 or IH_INLN_DALI_LIGHTLEVEL.MAX_LEVEL)
INPUT_INVALID_ERROR	16#3017	16#0004	The ‘Address’ input is not valid For Group (0< Address >15) For Ballast (0< Address >63) For Query (0 < Address >63 and <> 255
INPUT_INVALID_ERROR	16#3017	16#0005	The ‘DimMode’ input is not valid
INPUT_INVALID_ERROR	16#3017	16#0006	The ‘AddressType’ input is not valid
INPUT_INVALID_ERROR	16#3017	16#0007	The ‘Scene’ input is not valid
INPUT_INVALID_ERROR	16#3017	16#0008	SizeOfActualLevel is not valid(1< SizeOfActualLevel > 64)
INPUT_INVALID_ERROR	16#3017	16#0009	SizeOfStatus is not valid (1<SizeOfStatus> 64)
DEVICE_ERROR	16#3021	16#0000	Time out error. The device is not Responding
DEVICE_ERROR	16#3021	16#0001	An Invalid response received from the ballast (Dali bus error)
DEVICE_ERROR	16#3021	16#0004	Unsupported command has been received via inline (no action on the Dali bus)
DEVICE_ERROR	16#3021	16#0008	I/O error (DALI supply failure)
DEVICE_ERROR	16#3021	16#000F	Undefined error
RESOURCE_ERROR	16#3018	16#0000	Zero pointer, variable not assigned
OTHER_ERROR	16#3031	16#0000	Invalid token or token not available between IH_DaliReadWrite and other Dali blocks

Tab. 19-80: RIH_Dali error codes

20 RIH_Modbus.library

20.1 RIH_Modbus, general information

The library **RIH_Modbus** is to be used with the Inline modules **R-IB IL RS232 PRO**, **R-IB IL RS485/422 PRO** or **R-IB IL RS-Uni** (Inline modules for a serial data transfer). The Inline modules can be connected to the control or an Inline bus coupler (Profibus, Profinet, Sercos).



(1) 1..n Serial communication modules
(2) 1..x Devices

Fig. 20-1: Modbus RTU network example with Lx5 control, R-IB IL RS485/422-PRO and SEFC3600 frequency converter

The **RIH_Modbus** library provides function blocks to communicate with the Modbus RTU protocol as master or client.

To start in PLC program with RIH_Modbus, the function block **IH_MBRTUSlave**has to be instantiated for the client and the **IH_MBRTUMaster** for the master.

The function block

- **IH_MBRTUMaster** provides the communication handle for each other function blocks in the library RIH_Modbus, e.g. IH_MBReadCoils and IH_MBWriteSingleCoil.
- **IH_MBRTUSlave** ensures the communication with a Modbus RTU master.

With the Inline modules **R-IB IL RS232 PRO**, **R-IB IL RS485/422 PRO** or **R-IB IL RS-Uni**, a direct data exchange between two Bosch Rexroth Lx5 controls is possible using **IH_MBRTUMaster** and **IH_MBRTUSlave**.

Each function block with Modbus function code can be used in a single request or in the polling mode. In polling mode, the polling interval can be specified and the function block sends the requests automatically on time.

Please pay attention if Modbus RTU with EFC frequency converters are used.

Currently, one Modbus RTU connection requires approx. 100 ms for sending and receiving. Thus, 10 requests take 1 second.

Example: 5 Modbus RTU devices, 2 requests to each device = 10 requests.

If high update rates are expected or many Modbus RTU devices are used, additional serial Inline communication modules are required.

Other Modbus RTU applications might need less than 100 ms.

Function blocks of the RIH_Modbus library

Type	Description	Function code
IH_MBGetCommEventCounter, page 787	Returns the status word and the communication event counter	11
IH_MBGetCommEventLog, page 790	Returns the event log structure which includes the status word, communication event counter, message counter and a field of event bytes	12
IH_MBMaskWriteRegister, page 794	Modifies the content of a specified holding register using a combination of an AND mask, an OR mask, and the register's current content	22
IH_MBReadCoils, page 797	Reads the ON or OFF value from 1 to 2000 coils, i.e. continuous output bits	1
IH_MBReadDeviceIdentification, page 801	Reads the Identification and additional information related to the physical and functional description of a remote device from a client (master)	43
IH_MBReadDiagnostics, page 807	Provides tests to check the communication between master and slave or internally in the master	8
IH_MBReadDiscreteInputs, page 810	Read from 1 to 2000 neighboring status of discrete inputs, bit data access	2
IH_MBReadExceptionStatus, page 814	Reads the diagnostic status byte (8 bits) of the Modbus RTU slave	7
IH_MBReadFIFOQue, page 816	Reads the content of a First-In-First-Out (FIFO) queue of register of a remote device	24
IH_MBReadFileRecord, page 819	Reads the specified file information, subrequests are possible	20
IH_MBReadHoldingRegisters, page 823	Reads the content of a neighboring holding register block, word data access	3
IH_MBReadInputRegisters, page 827	Reads from 1 to 125 neighboring input registers, word data access	4
IH_MBReadWriteMultipleRegisters, page 831	Combine a read and a write operation in one Modbus transaction. The write operation is performed before the reading	23
IH_MBRTUMaster, page 835	Provides the communication handle for all other function blocks in the library RIH_Modbus, e.g. IH_MBReadCoils, IH_MBWriteSingleCoil.	

Type	Description	Function code
IH_MBRTUSlave, page 840	Ensures the communication with a Modbus RTU master, e.g. can be used to exchange data with the RIH_MBRTU-Master	
IH_MBWriteFileRecord, page 846	Writes the specified file information, subrequests are possible	21
IH_MBWriteMultipleCoils, page 851	Writes multiple ON or OFF values from 1 to 1968 coils, i.e. continuous output bits	15
IH_MBWriteMultipleRegisters, page 854	Writes multiple registers (1 to 123 registers), word data access	16
IH_MBWriteSingleCoil, page 858	Write an ON or OFF value to a single coil, i.e. single output bit	5
IH_MBWriteSingleRegister, page 861	Writes a single holding register, word data access	6

Tab. 20-1: Function block of the RIH_Modbus library

20.2 IH_MBGetCommEventCounter

Function code FC 11

Brief description The function block **IH_MBGetCommEventCounter** is used to get a status word and an event count from the communication event counter of the slave device. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-2: Reference table

Interface description

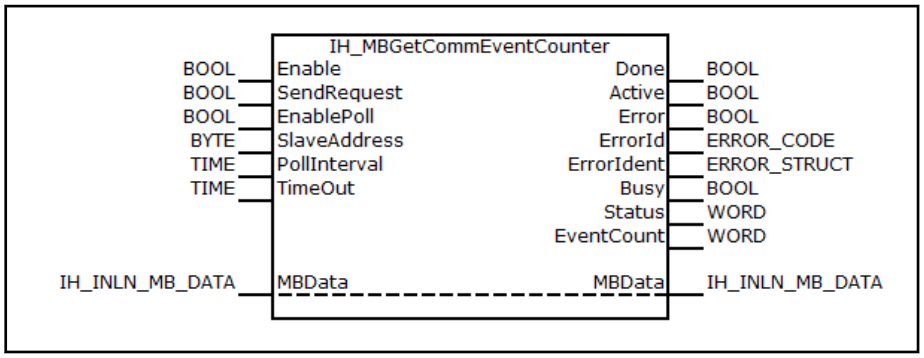


Fig. 20-2: IH_MBGetCommEventCounter function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
	Status	WORD	Status word
	EventCount	WORD	Communication event count
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of the function codes IH_INLN_MB_DATA, page 864

Tab. 20-3: Interface variables of the IH_MBGetCommEventCounter function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-4: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBGetCommEventCounter

Functional description

The function block **IH_MBGetCommEventCounter** (FC 11) is used to get a status word and an event count from the communication event counter of the slave device. The function block should be called with the Modbus master

function block. "MBData" should link to the master function block in the program; its structure communicates and exchanges the data.

The device's event counter is incremented once for each successful message completion. It is not incremented for exception responses, poll commands, or gets event counter commands.

The normal response contains a two–byte status word, and a two–byte event count. The status word will be all ones (FF FF hex), if a previously issued program command is still being processed by the remote device (a busy condition exists). Otherwise, the status word will be all zeros.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the **Done** bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The function block IH_MBGetCommEventCounter should be instantiated along with IH_MBRTUMaster to communicate with the Modbus slave. Use a common variable at the MBData of the IH_ MBGetCommEventCounter and IH_MBRTUMaster function blocks to exchange the data with the Modbus slave.

In this example, the function block IH_ MBGetCommEventCounter is used to get the status word and event count of slave with address 1.

Enable both the IH_MBRTUMaster and IH_MBGetCommEventCounter function block, with the rising edge at the bSendRequest input, the function block reads the status word and event count of slave address 1 and the result is available at the wStatus and wEventCount

By setting the bEnablePoll variable to TRUE and Pollinterval to more than 1 second, the data is read from the slave with regular interval

Variable declaration

```
VAR
    fbIH_MBGetCommEventCounter: IH_MBGetCommEventCounter;
    bEnable: BOOL;
    bEnablePoll: BOOL;
    bSendRequest: BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    wStatus: WORD;
    wEventCount: WORD;
END_VAR
```

Implementation

```
fbIH_MBGetCommEventCounter(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= bEnablePoll,
    SlaveAddress:= BYTE#1,
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
```

```
Busy=> bBusy,
Status=> wStatus,
EventCount=> wEventCount);
```

20.3 IH_MBGetCommEventLog

Function code FC 12

Brief description The function block **IH_MBGetCommEventLog** is used to get a status word, event count, message count and field of event bytes in short communication event log from the slave module. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-5: Reference table

Interface description

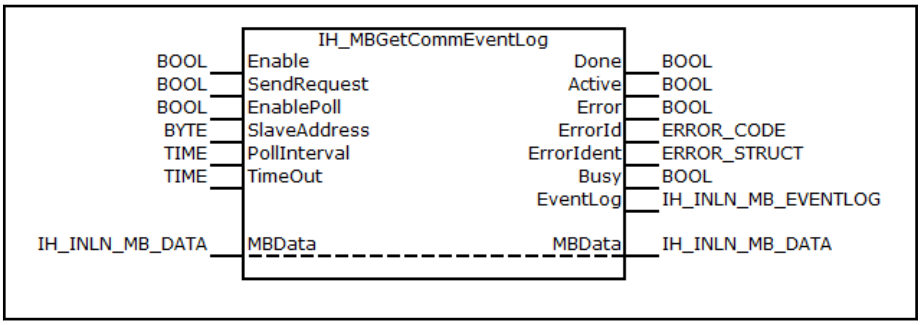


Fig. 20-3: IH_MBGetCommEventLog function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with the master
	EventLog	IH_INLN_MB_EVEN-TLOG	Structure gives additional information about Status, EventCount, MessageCount and Events IH_INLN_MB_EVENTLOG, page 867
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-6: Interface variables of the IH_MBGetCommEventLog function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-7: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBGetCommEventLog

Functional description

The function block **IH_MBGetCommEventLog** (FC 12) is used to get a status word, event count, message count and field of event bytes in a short communication event log from the slave module. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program; its structure communicates and exchanges the data.

The status word and event counts are identical to the data returned by the Get Communications Event Counter function (11, 16#0B).

The message counter contains the quantity of messages processed by the remote device since its last restart, clear counters operation, or power–up. The count is identical to the count returned by the diagnostic function (code 08), sub-function Return Bus Message Count (code 11, 0B hex).

The event bytes field contains 0-64 bytes, with each byte corresponding to the status of one MODBUS send or receive operation for the remote device. The remote device enters the events into the field in chronological order. Byte

0 is the most recent event. Each new byte overwrites the oldest byte from the field.

Event byte content

An event byte returned by the Get Communications Event Log function can be any of four types. The type is defined by bit 7 (the high–order bit) in each byte. It may be further defined by bit 6. This explained below.

Remote device MODBUS Receive Event

The remote device stores this type of event byte when a query message is received. It is stored before the remote device processes the message. The event is defined by bit 7 set to logic "1". The other bits will be set to logic "1" if the corresponding condition is TRUE. The bit layout is:

Bit contents

Bit	Description
0	Not Used
1	Communication Error
2	Not Used
3	Not Used
4	Character Overrun
5	Currently in Listen Only Mode
6	Broadcasts Received
7	1

Remote device MODBUS Send Event

The remote device stores this type of event byte when it finishes processing a request message. It is stored if the remote device returned a normal or exception response, or no response. The event is defined by bit 7 set to logic "0", with bit 6 set to a "1". The other bits will be set to logic "1" if the corresponding condition is TRUE. The bit layout is:

Bit contents

Bit	Description
0	Read Exception Sent (Exception Codes 1-3)
1	Slave Abort Exception Sent (Exception Code 4)
2	Slave Busy Exception Sent (Exception Codes 5-6)
3	Slave Program NAK Exception Sent (Exception Code 7)
4	Write Timeout Error Occurred
5	Currently in Listen Only Mode
6	1
7	0

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error in-

formation. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The function block **IH_MBGetCommEventLog** should be instantiated along with IH_MBRTUMaster to communicate with the Modbus slave. Use a common variable at the MBData of IH_ MBGetCommEventLog and IH_MBRTUMaster function blocks to exchange the data with the Modbus slave.

In this example, the function block IH_ MBGetCommEventLog is used to get a log of the slave with address 1.

Enable both the IH_MBRTUMaster and IH_ MBGetCommEventLog function block, with the rising edge at bSendRequest input the function block reads the status word and an event count of the slave address 1 and the result is available in the stEventLog.

By setting the bEnablePoll variable to TRUE and Pollinterval to more than 1 second, the data is read from the slave with regular interval.

After each successful completion of the read operation, the done bit is set to true for one PLC cycle

Variable declaration

```
VAR
    fbIH_MBGetCommEventLog: IH_MBGetCommEventLog;
    bEnable: BOOL;
    bSendRequest: BOOL;
    bEnablePoll: BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    stEventLog: IH_INLN_MB_EVENTLOG;
END_VAR
```

Implementation

```
fbIH_MBGetCommEventLog(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= bEnablePoll,
    SlaveAddress:= BYTE#1,
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    Eventlog=> stEventLog);
```

Response of event log

In this example, the status word is 16#0000, indicating that the remote device is not processing a program function. The event count shows that 264 (16#0108) events have been counted by the remote device. The message count shows that 289 (16#0121) messages have been processed.

The most recent communications event is shown in the Event 0 byte. Its contents 32 (16#20) show that the remote device has most recently entered the Listen Only Mode.

The previous event is shown in the Event 1 byte. Its contents (00 hex) show that the remote device received a Communications Restart.

Expression	Type	Value
stEventLog	IH_INLN_MB_EVENTLOG	
Status	WORD	0
EventCount	WORD	264
MessageCount	WORD	289
Events	ARRAY [0..63] OF BYTE	
Events[0]	BYTE	32
Events[1]	BYTE	0

Fig. 20-4: stEventLog

20.4 IH_MBMaskWriteRegister

Function code FC 22

Brief description The function block **IH_MBMaskWriteRegister** is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the registers of the slave module. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-8: Reference table

Interface description

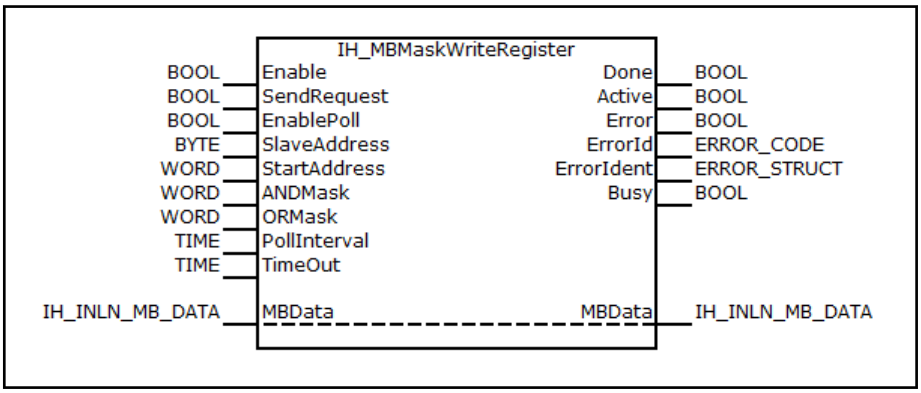


Fig. 20-5: IH_MBMaskWriteRegister function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of holding register
	ANDMask	WORD	AND mask value
	ORMask	WORD	OR mask value
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-9: Interface variables of the IH_MBMaskWriteRegister function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	65535	0	Rising edge at Enable
ANDMask	WORD	0	65535	0	Rising edge at Enable
ORMask	WORD	0	65535	0	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-10: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBMaskWriteRegister

Functional description

The function block **IH_MBMaskWriteRegister** (FC22) is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the registers of the slave module. The function can be used to set or clear individual bits in the register. The function block should be called with the Modbus master function block. "MBData" should link to the master function block to communicate and exchanges the data with the master.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the IH_MBMaskWriteRegister function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBMaskWriteRegister function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBMaskWriteRegister function block is used to mask a register in the slave device.

Mask register with SendRequest condition

The **IH_MBMaskWriteRegister** is used to mask a register of address 100 to slave Address 2 using the "bSendRequest" as BOOL to trigger the mask register.

Variable declaration

```
VAR
    fbIH_MBMaskWriteRegister: IH_MBMaskWriteRegister;
    bEnable: BOOL;
    bSendRequest: BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBMaskWriteRegister(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#100,
    ANDMask:= WORD#16#F2,
    ORMask:= WORD#16#25,
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

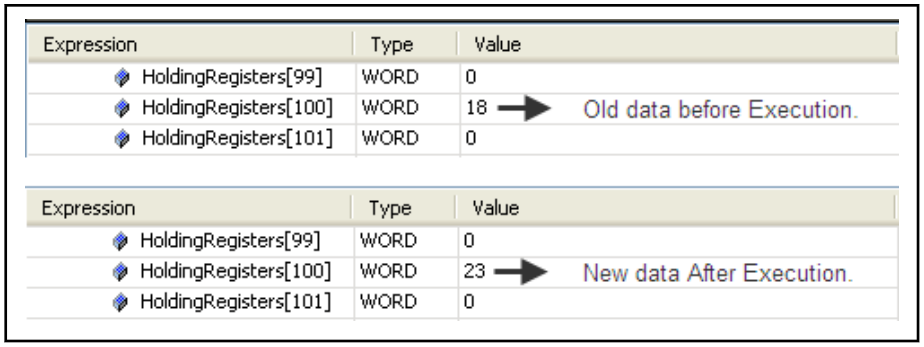


Fig. 20-6: Shows the modification after the execution in the slave device

20.5 IH_MBReadCoils

Function code FC 1

Brief description The function block **IH_MBReadCoils** is used to read ON or OFF statuses of coils, i.e. output bits from the slave module. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-11: Reference table

Interface description

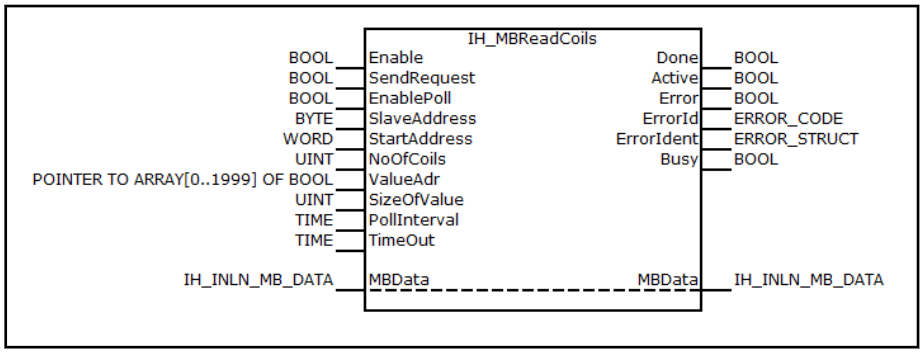


Fig. 20-7: IH_MBReadCoils function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of output bits
	NoOfCoils	UINT	Quantity of coils (discrete outputs)
	ValueAdr	POINTER TO ARRAY[0..1999] OF BOOL	Address of the Array to store read value of the output bits. Use ADR () operator
	SizeOfValue	UINT	Size of the Value array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-12: Interface variables of the IH_MBReadCoils function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	65535	0	Rising edge at Enable
NoOfCoils	UINT	1	2000	1	Rising edge at Enable
SizeOfValue	UINT	1	2000	----------	Rising edge at Enable

Name	Type	Min. value	Max. value	Default value	Effective
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-13: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadCoils

Functional description

The function block **IH_MBReadCoils** (FC 1) is used to read the status of 1 to 2000 coils, i.e. output bits continuous in sequence from the slave module. The function block should be called with the Modbus master function block. ‘MBData’ should link to the master function block in the program; its structure communicates and exchanges the data.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the IH_MBReadCoils function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBReadCoils function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBReadCoils function block is used to read coils from the slave device.

Read coils with SendRequest condition

The **IH_MBReadCoils** is used to read 8 coils of the Start address 0x1000 from slave Address 2 using the "bSendRequest" as BOOL to trigger the read coils.

Variable declaration

```
VAR
    fbIH_MBReadCoils: IH_MBReadCoils;
    bEnable: BOOL;
    bSendRequest: BOOL;
    arReadCoils: ARRAY [0..20] OF BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadCoils(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#1000,
    NoOfCoils:= UINT#8,
    ValueAdr:= ADR(arReadCoils),
    SizeOfValue:= SIZEOF(arReadCoils),
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
```

```
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

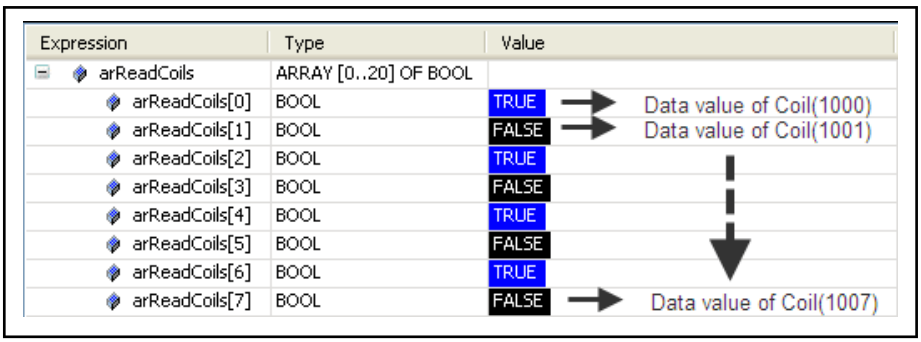


Fig. 20-8: arReadCoils' for program 1

Read coils with Poll interval condition

IH_MBReadCoils is used to read 5 coils of the Start address 0x0500 from slave Address 5 using the "Poll Interval" (3 seconds) to trigger the reading coils.

Variable declaration

```
VAR
    fbIH_MBReadCoils2: IH_MBReadCoils;
    bEnable2: BOOL;
    arReadCoils2: ARRAY [0..4] OF BOOL;
 stMBData: IH_INLN_MB_DATA;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
    stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadCoils2(
    Enable:= bEnable2,
    SendRequest:= FALSE,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#5,
    StartAddress:= WORD#500,
    NoOfCoils:= UINT#5,
    ValueAdr:= ADR(arReadCoils2),
    SizeOfValue:= SIZEOF(arReadCoils2),
    PollInterval:= T#3S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
    ErrorID=> enErrorID2,
    ErrorIdent=> stErrorIdent2,
    Busy=> bBusy2);
```

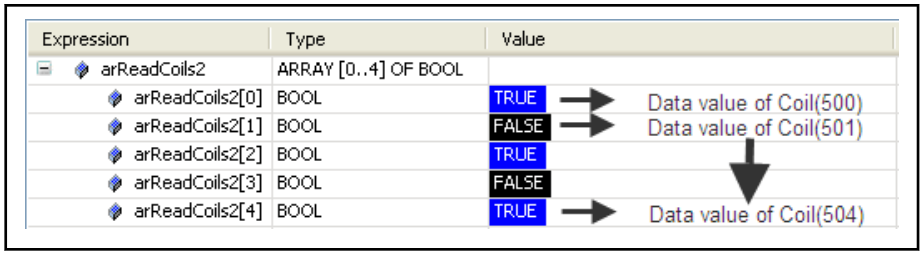


Fig. 20-9: arReadCoils2' for program 2

20.6 IH_MBReadDeviceIdentification

Function code FC 43

Brief description The function block **IH_MBReadDeviceIdentification** uses FC 43(0x2B) with MEI (MODBUS Encapsulated Interface) type 14 (0x0E) to read the identification and additional information related to the physical and functional description of a remote device from a client (Master). The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-14: Reference table

Interface description

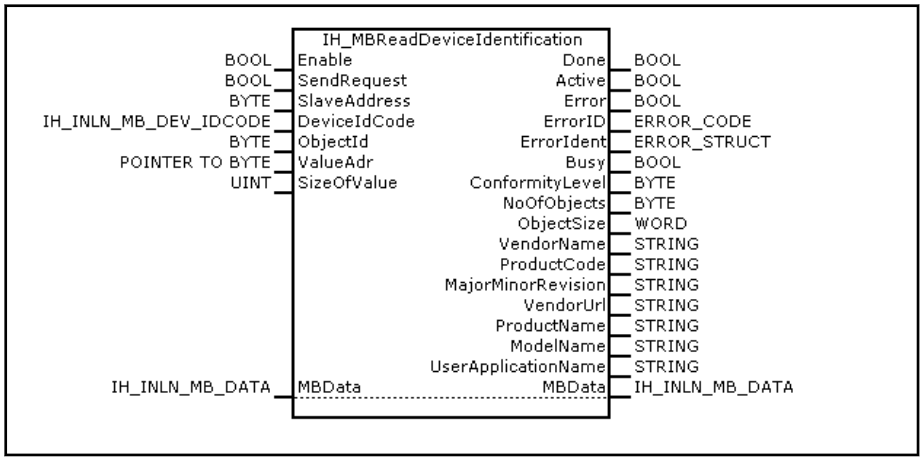


Fig. 20-10: IH_MBReadDeviceIdentification function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	SlaveAddress	BYTE	Slave address
	DeviceIdCode	IH_INLN_MB_DEV_ID-CODE	Parameter to define access types for available Object Id's (Read Device ID Code) IH_INLN_MB_DEV_IDCODE, page 864
	ObjectId	BYTE	Data object identification (Object Id)
	ValueAdr	POINTER TO BYTE	Address of Object

I/O type	Name	Data type	Description
	SizeOfValue	UINT	Size of Object
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
	ConformityLevel	BYTE	Identification conformity level of the device and type of supported access
	NoOfObjects	BYTE	No. of. Identification objects returned in the response
	ObjectSize	WORD	Total data size of requested 'DeviceIdCode' or 'ObjectId' elements in bytes ObjectSize = (ObjectId(1byte)+Object length(1byte)+ Value in Objectlength) * NoOfObjects
	VendorName	STRING	'VendorName' returned from Object Id. : 0x00
	ProductCode	STRING	'ProductCode' returned from Object Id. : 0x01
	MajorMinorRevision	STRING	'MajorMinorRevision' returned from Object Id. : 0x02
	VendorUrl	STRING	'VendorUrl' returned from Object Id. : 0x03
	ProductName	STRING	'ProductName' returned from Object Id. : 0x04
	ModelName	STRING	'ModelName' returned from Object Id. : 0x05
	UserApplicationName	STRING	'UserApplicationName' returned from Object Id. : 0x06
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-15: Interface variables of the IH_MBReadDeviceIdentification function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. Value	Default Value	Application
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with "Enable"
SlaveAddress	BYTE	0	247	0	Rising Edge at "Enable"
DeviceIdCode	IH_INLN_MB_DEV_ID-CODE	1	4	1	Rising Edge at "Enable"
ObjectId	BYTE	16#00	16#FF	16#00	Rising Edge at "Enable"
ValueAdr	POINTER TO BYTE	n.def.	n.def.	0	Continuous with "Enable"

Name	Type	Min. value	Max. Value	Default Value	Application
SizeOfValue	UINT	0	Depends on Slave device object size	0	Rising Edge at "Enable"
Timeout	TIME	T#0ms	T#3600s	T#1s	Rising Edge at "Enable"

Tab. 20-16: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadDeviceIdentification

Functional description

The function block **IH_MBReadDeviceIdentification** uses FC 43(0x2B) with MEI (MODBUS Encapsulated Interface) type 14 (0x0E) to read the identification and additional information related to the physical and functional description of a remote device from a client (Master).

Reading of device identification is done from set of addressable data objects. Data objects are addressed using object id. Function block allows to access following categories of data objects:

- **Basic Device Identification:** All objects of this category are mandatory: VendorName, Product code, and revision number.
- **Regular Device Identification:** All objects of this category are defined in the standard but implementation is optional.
- **Extended Device Identification:** It provides the additional and optional identification and private description data about physical device itself. All of these are device dependent and optional.

Supported data objects with object Id and categories are shown in the following table.

Object ID	Object Name / Description	Type	M / O	category
16#00	VendorName	STRING	**Mandatory**	**Basic**
16#01	ProductCode	STRING	**Mandatory**	
16#02	MajorMinorRevision	STRING	**Mandatory**	
16#03	VendorUrl	STRING	Optional	**Regular**
16#04	ProductName	STRING	Optional	
16#05	ModelName	STRING	Optional	
16#06	UserApplicationName	STRING	Optional	
16#07 ... 16#7F	reserved		Optional	
16#80 ... 16#FF	Private objects may be optionally defined The range [16#80 - 16#FF] is product dependant	device dependant	Optional	**Extended**

Object category access levels are specified in Input "DeviceIdCode". Function block supports the following Read Device Id code access types:

- 01: **IH_INLN_MB_READ01** - request to get the basic device identification (stream access)
- 02: **IH_INLN_MB_READ02** - request to get the regular device identification (stream access)

- 03: **IH_INLN_MB_READ03** - request to get the extended device identification (stream access)
- 04: **IH_INLN_MB_READ04** - request to get one specific identification object (individual access)

For an individual object id access "DeviceIdCode" as **IH_INLN_MB_READ04**, desired object id should be specified in input "ObjectId". For the rest of device id codes, "ObjectId" should be 0.

Objects with defined data type in Basic and Regular categories are processed as direct function block outputs i.e. "VendorName", "ProductCode" … "UserApplicationName". All other device dependant objects are processed as Byte data type in default to "ValueAdr".

For the ObjectId greater than 7, the read data is available at "ValueAdr". The "ValueAdr" contains the ObjectID, ObjectSize and Object value.

If multiple Objects are available then the data is stored in the following format and the valid data size is available at "ObjectSize".

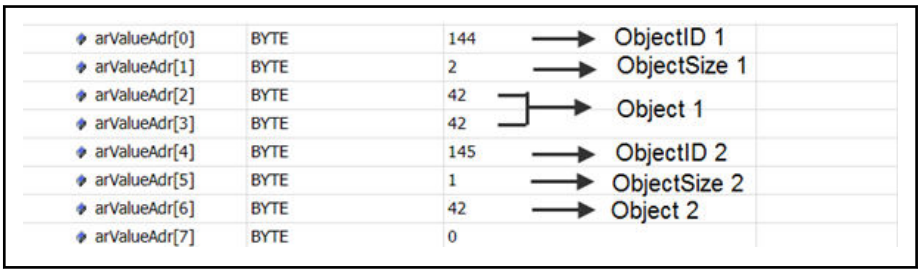


Fig. 20-11: Multiple Objects

Output "ConformityLevel" states actual Identification conformity level from the device and type of supported Access for the requested inputs "objectId" and "DeviceIdCode". Following information will be response in "ConformityLevel":

- 0x01: basic identification (stream access only)
- 0x02: regular identification (stream access only)
- 0x03: extended identification (stream access only)
- 0x81: basic identification (stream access and individual access)
- 0x82: regular identification (stream access and individual access)
- 0x83: extended identification(stream access and individual access)

For a requested "DeviceIdCode", output "NoOfObjects" states no. of. Identification objects returned in response.

For `DeviceIdCode = IH_INLN_MB_READ04`, "NoOfObjects" will always be 1.

The size of Object Id's from 0x07…0xFF need to be identified from the physical device datasheet and assign the required length to fill the data.

This function block should be called with MODBUS master function block. "MBData" should link to master function block in the program, this is a common structure to communicate and exchange the data between these two function blocks.

The rising edge at Enable input starts the operation and after successful completion the output "Done" is set to TRUE. If any error then the Error bit is set and the detailed error information is available at ErrorID and ErrorIdent.

Handling Specials

This Function block is tested with CIMS sensor only and not with any other devices supporting Function Code 43.

Though the function block supports Read Device ID Code from **IH_INLN_MB_READ01 to IH_INLN_MB_READ04**, CIMS sensor supports only **IH_INLN_MB_READ04** testing was not done for other device id codes.

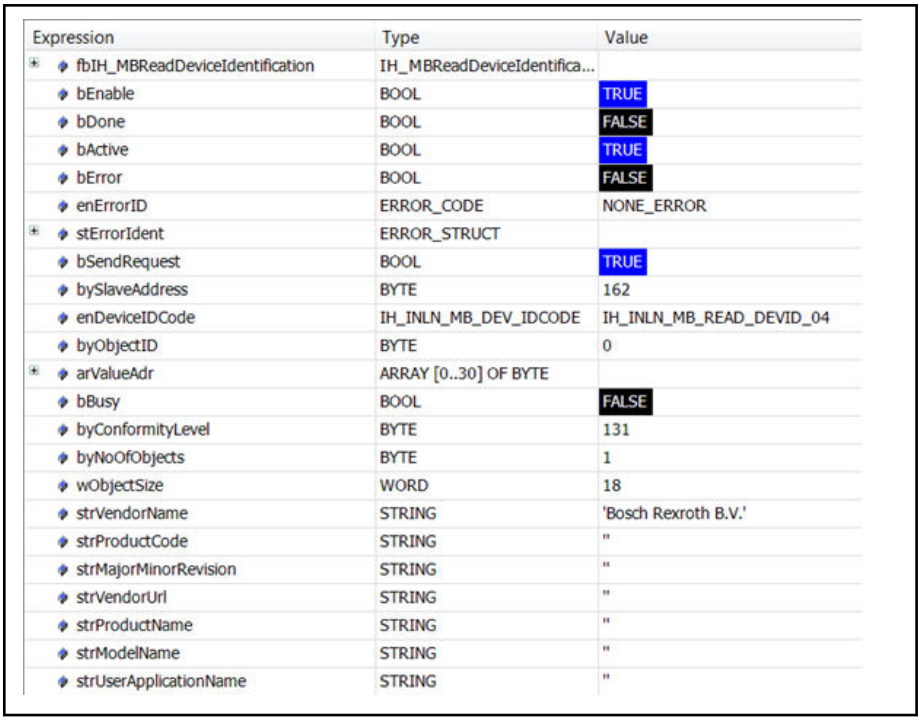
Example1: Communication with the Bosch Rexroth Sensor

Declaration

```
VAR
    fbIH_MBReadDeviceIdentification: IH_MBReadDeviceIdentification;
    bEnable: BOOL;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bSendRequest: BOOL;
    bySlaveAddress: BYTE := 162;
    enDeviceIDCode: IH_INLN_MB_DEV_IDCODE := IH_INLN_MB_DEV_IDCODE.IH_INLN_MB_READ_DEVID_04;
    byObjectID: BYTE := 0;
    arValueAdr: array[0..30] of byte;
    bBusy: BOOL;
    byConformityLevel: BYTE;
    byNoOfObjects: BYTE;
    wObjectSize: WORD;
    strVendorName: STRING;
    strProductCode: STRING;
    strMajorMinorRevision: STRING;
    strVendorUrl: STRING;
    strProductName: STRING;
    strModelName: STRING;
    strUserApplicationName: STRING;
END_VAR
```

Implementation

```
fbIH_MBReadDeviceIdentification(
    Enable:= bEnable,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    SendRequest:= bSendRequest,
    SlaveAddress:= bySlaveAddress,
    DeviceIdCode:= enDeviceIDCode,
    ObjectId:= byObjectID,
    ValueAdr:= ADR(arValueAdr),
    SizeofValue:= SIZEOF(arValueAdr),
    TimeOut:= T#2S,
    Busy=> bBusy,
    ConformityLevel=> byConformityLevel,
    NoOfObjects=> byNoOfObjects,
    ObjectSize=> wObjectSize,
    VendorName=> strVendorName,
    ProductCode=> strProductCode,
    MajorMinorRevision=> strMajorMinorRevision,
    VendorUrl=> strVendorUrl,
    ProductName=> strProductName,
    ModelName=> strModelName,
    UserApplicationName=> strUserApplicationName,
    MBData:= stMBData);
```



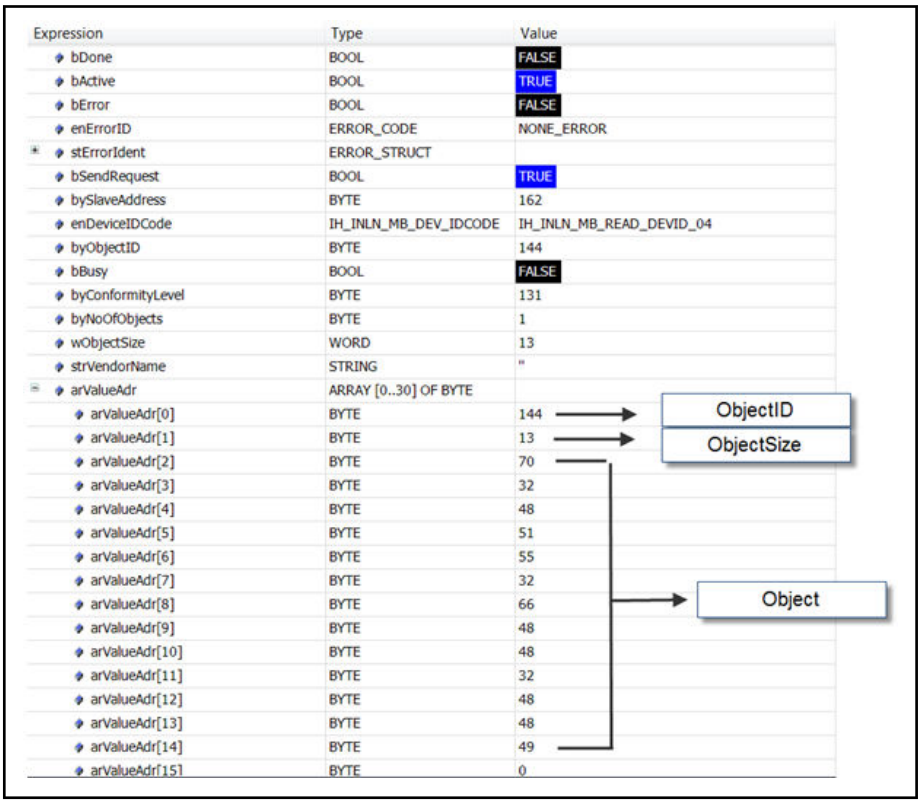
Expression	Type	Value
fbIH_MBReadDeviceIdentification	IH_MBReadDeviceIdentifica...	
bEnable	BOOL	TRUE
bDone	BOOL	FALSE
bActive	BOOL	TRUE
bError	BOOL	FALSE
enErrorID	ERROR_CODE	NONE_ERROR
stErrorIdent	ERROR_STRUCT	
bSendRequest	BOOL	TRUE
bySlaveAddress	BYTE	162
enDeviceIDCode	IH_INLN_MB_DEV_IDCODE	IH_INLN_MB_READ_DEVID_04
byObjectID	BYTE	0
arValueAdr	ARRAY [0..30] OF BYTE	
bBusy	BOOL	FALSE
byConformityLevel	BYTE	131
byNoOfObjects	BYTE	1
wObjectSize	WORD	18
strVendorName	STRING	'Bosch Rexroth B.V.'
strProductCode	STRING	''
strMajorMinorRevision	STRING	''
strVendorUrl	STRING	''
strProductName	STRING	''
strModelName	STRING	''
strUserApplicationName	STRING	''

Fig. 20-12: Result1

Example2:

Declaration

```
VAR
byObjectID: BYTE := 16#90; // Read order number in CIMS Sensor.
END_VAR
```



Expression	Type	Value
bDone	BOOL	FALSE
bActive	BOOL	TRUE
bError	BOOL	FALSE
enErrorID	ERROR_CODE	NONE_ERROR
stErrorIdent	ERROR_STRUCT	
bSendRequest	BOOL	TRUE
bySlaveAddress	BYTE	162
enDeviceIDCode	IH_INLN_MB_DEV_IDCODE	IH_INLN_MB_READ_DEVID_04
byObjectID	BYTE	144
bBusy	BOOL	FALSE
byConformityLevel	BYTE	131
byNoOfObjects	BYTE	1
wObjectSize	WORD	13
strVendorName	STRING	''
arValueAdr	ARRAY [0..30] OF BYTE	
arValueAdr[0]	BYTE	144
arValueAdr[1]	BYTE	13
arValueAdr[2]	BYTE	70
arValueAdr[3]	BYTE	32
arValueAdr[4]	BYTE	48
arValueAdr[5]	BYTE	51
arValueAdr[6]	BYTE	55
arValueAdr[7]	BYTE	32
arValueAdr[8]	BYTE	66
arValueAdr[9]	BYTE	48
arValueAdr[10]	BYTE	48
arValueAdr[11]	BYTE	32
arValueAdr[12]	BYTE	48
arValueAdr[13]	BYTE	48
arValueAdr[14]	BYTE	49
arValueAdr[15]	BYTE	0

Fig. 20-13: Result2

20.7 IH_MBReadDiagnostics

Function code FC 8

Brief description The function block **IH_MBReadDiagnostics** provides a series of tests for checking the communication system between a client (Master) device and a server (Slave), or for checking various internal error conditions within a server in RTU mode. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-17: Reference table

Interface description

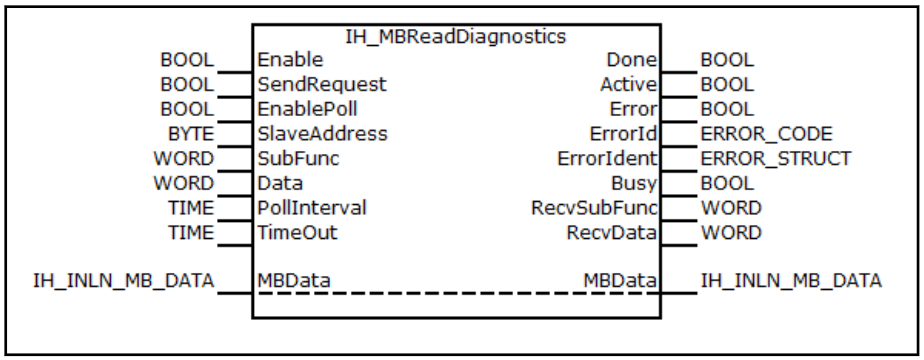


Fig. 20-14: IH_MBReadDiagnostics function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	SubFunc	WORD	Sub-function code
	Data	WORD	Data field
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
	RecvSubFunc	WORD	Received subfunction from device
	RecvData	WORD	Response data from device
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-18: Interface variables of the IH_MBReadDiagnostics function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-19: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadDiagnostics

Functional description

The function block **IH_MBReadDiagnostics** provides a series of tests for checking the communication system between a client (Master) device and a server (Slave), or for checking various internal error conditions within a server. The function uses a two–byte sub-function code field in the query to define the type of test to be performed. The server echoes both the function code and sub-function code in a normal response. Some of the diagnostics cause data to be returned from the remote device in the data field of a normal response. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program, its structure communicates and exchanges the data.

Subfunction codes supported by the serial line devices:

Diagnostic name	SubFunction code	Data (Requested)	Data (Response)
Return Query Data	16#0000	ANY	Echo request data
Restart Communications Option	16#0001	16#0000 (or) 16#FF00	Echo request data

Diagnostic name	SubFunction code	Data (Requested)	Data (Response)
Return Diagnostic Register	16#0002	16#0000	Diagnostic register contents
Change ASCII Input Delimiter	16#0003	CHAR 00	Echo request data
Force Listen Only Mode	16#0004	16#0000	No response returned
RESERVED	16#05 … 16#09	----------	----------
Clear Counters and Diagnostic Register	16#000A	16#0000	Echo request data
Return Bus Message Count	16#000B	16#0000	Total message count
Return Bus Communication Error Count	16#000C	16#0000	CRC error count
Return Bus Exception Error Count	16#000D	16#0000	Exception error count
Return Slave Message Count	16#000E	16#0000	Slave message count
Return Slave No Response Count	16#000F	16#0000	Slave no response count
Return Slave NAK Count	16#0010	16#0000	Slave NAK count
Return Slave Busy Count	16#0011	16#0000	Slave device busy count
Return Bus Character Overrun Count	16#0012	16#0000	Slave character overrun count
RESERVED	16# 0013	----------	----------
Clear Overrun Counter and Flag	16#0014	16#0000	Echo request data
RESERVED	N.A	----------	----------

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the ‘Enable’ input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the IH_MBReadDiagnostics function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBReadDiagnostics function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBReadDiagnostics function block is used to read the diagnostic status of the slave device.

Reading diagnostics with SendRequest condition

The **IH_MBReadDiagnostics** is used to read (Return Diagnostic Register) from the slave Address 2.

Variable declaration

```
VAR
    fbIH_MBReadDiagnostics: IH_MBReadDiagnostics;
    bEnable: BOOL;
```

```
    bSendRequest: BOOL;
 wSubFunc : word;
 wData : word;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    wRecvSubFunc: WORD;
    wRecvdata: WORD;
END_VAR
```

Implementation

```
fbIH_MBReadDiagnostics(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    SubFunc:= wSubFunc,
    Data:= wData,
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    RecvSubFunc=> wRecvSubFunc,
    RecvData=> wRecvdata);
```

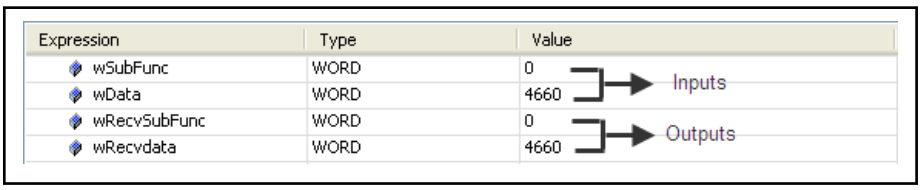


Fig. 20-15: Response for the program

20.8 IH_MBReadDiscreteInputs

Function code FC 2

Brief description The function block **IH_MBReadDiscreteInputs** is used to read input bits in RTU mode. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-20: Reference table

Interface description

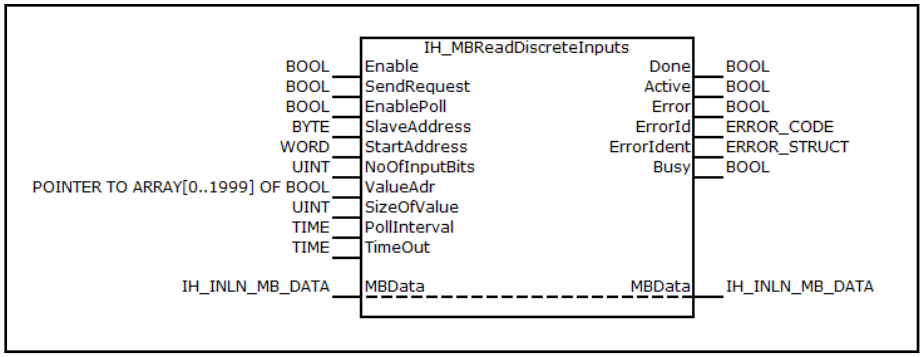


Fig. 20-16: IH_MBReadDiscreteInputs function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for **SendRequest** to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of input bits
	NoOfInputBits	UINT	Quantity of input bits to read
	ValueAdr	POINTER TO ARRAY[0..1999] OF BOOL	Address of the Array to store input bits. Use ADR () operator
	SizeOfValue	UINT	Size of the Value array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-21: Interface variables of the IH_MBReadDiscreteInputs function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	16#FFFF	0	Rising edge at Enable
NoOfInputBits	UINT	1	2000	1	Rising edge at Enable
SizeOfValue	UINT	1	2000	----------	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-22: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadDiscreteInputs

Functional description

The function block **IH_MBReadDiscreteInputs** (FC 2) is used to read input bits in RTU mode. The function block should be called with the Modbus master function block IH_MBRTUMaster. "MBData" should link to the master function block in the program, its structure communicates and exchanges the data.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

IH_MBRTUMaster function block should be instantiated along with the IH_MBReadDiscreteInputs function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between thethe IH_MBRTUMaster and IH_MBReadDiscreteInputs function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBReadDiscreteInputs function block is used to read the discrete inputs from the slave device.

Reading discrete inputs with SendRequest condition

The IH_MBReadDiscreteInputs is used to read 10 discrete inputs of the Start address 0x1000 from slave Address 2 using "bSendRequest" as BOOL to trigger the Read inputs.

Variable declaration

```
VAR
    fbIH_MBReadDiscreteInputs: IH_MBReadDiscreteInputs;
    bEnable: BOOL;
    bSendRequest: BOOL;
    arDiscreteInputs: ARRAY [0..2000] OF BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
```

```
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadDiscreteInputs(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#1000,
    ValueAdr:= ADR(arDiscreteInputs),
    NoOfInputBits:= UINT#10,
    SizeOfValue:= SIZEOF(arDiscreteInputs),
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

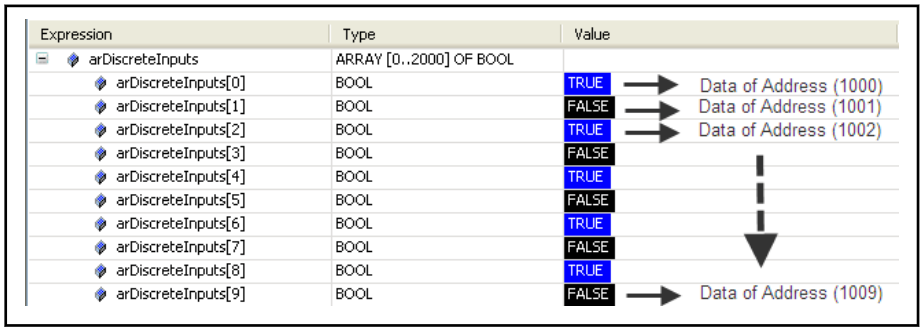


Fig. 20-17: arReadDiscreteInputs for program 1

Reading discrete inputs with Poll interval condition

The IH_MBReadDiscreteInputs is used to read 5 discrete inputs of the Start address 0x0500 from slave Address 5 using Poll interval (3 seconds) to trigger the Read inputs.

Variable declaration

```
VAR
    fbIH_MBReadDiscreteInputs2: IH_MBReadDiscreteInputs;
    bEnable2: BOOL;
    arDiscreteInputs2: ARRAY [0..10] OF BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
    stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadDiscreteInputs2(
    Enable:= bEnable2,
    SendRequest:= FALSE,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#5,
    StartAddress:= WORD#500,
    ValueAdr:= ADR(arDiscreteInputs2),
    NoOfInputBits:= UINT#5,
    SizeOfValue:= SIZEOF(arDiscreteInputs2),
    PollInterval:= T#3S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone2,
```

```
Active=> bActive2,
Error=> bError2,
ErrorID=> enErrorID2,
ErrorIdent=> stErrorIdent2,
Busy=> bBusy2);
```

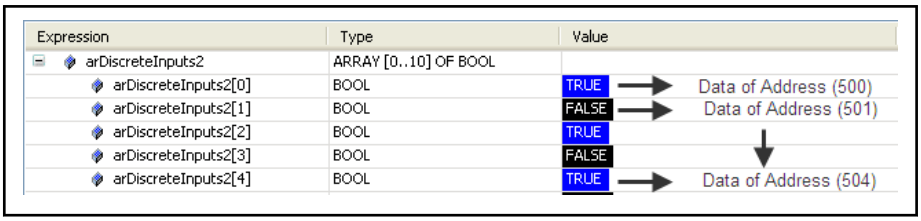


Fig. 20-18: arDiscreteInputs2 for program 2

20.9 IH_MBReadExceptionStatus

Function code FC 7

Brief description The function block **IH_MBReadExceptionStatus** is used to read the status of the Exception output bits in a slave device using the RTU mode. The function block should be called with the Modbus master function block IH_MBRTU-Master, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-23: Reference table

Interface description

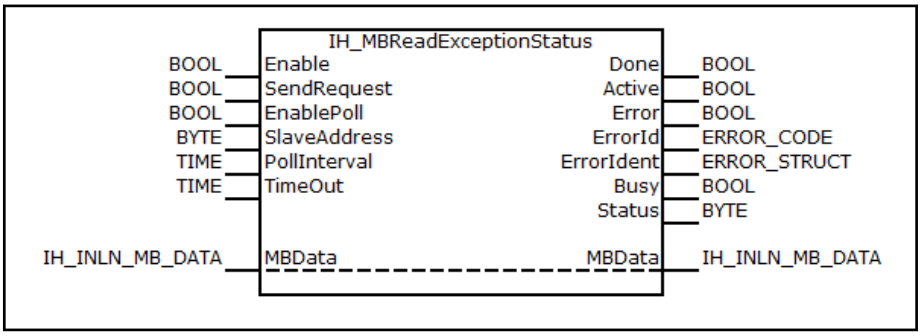


Fig. 20-19: IH_MBReadExceptionStatus function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for **SendRequest** to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
	Status	BYTE	It contains eight exception statuses in output bits of remote device
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-24: Interface variables of the IH_MBReadExceptionStatus function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-25: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadExceptionStatus

Functional description

The function block **IH_MBReadExceptionStatus** (FC 7) is used to read the status of the Exception output bits in a slave device using RTU mode. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program, its structure communicates and exchanges the data.

It reads the contents of eight exception status coils within the slave controller. Certain coils have predefined assignments in the various controllers. Other coils can be programmed by the user to hold information about the controller's status, for example, "machine ON/OFF", "heads retracted", "safeties satisfied", "error conditions exist", or other user–defined flags.

The exception coil assignments are predefined and depend on the controller model used.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error in-

formation. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block:

The IH_MBRTUMaster function block should be instantiated along with the IH_MBReadExceptionStatus function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBReadExceptionStatus function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBReadExceptionStatus function block is used to read the exception status from the slave device.

Reading Status byte with SendRequest condition

The **IH_MBReadExceptionStatus** is used to read the Status byte from slave Address 2, using "bSendRequest" as BOOL to trigger the Reading of Exception status.

Variable declaration

```
VAR
    fbIH_MBReadExceptionStatus: IH_MBReadExceptionStatus;
    bEnable: BOOL;
    bSendRequest: BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    byStatus: BYTE;
END_VAR
```

Implementation

```
fbIH_MBReadExceptionStatus(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    Status=> byStatus);
```

20.10 IH_MBReadFIFOQue

Function code FC 24

Brief description The function block **IH_MBReadFIFOQue** is used to read First-In-First-Out, i.e. the FIFO queue of register of the slave module. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-26: Reference table

Interface description

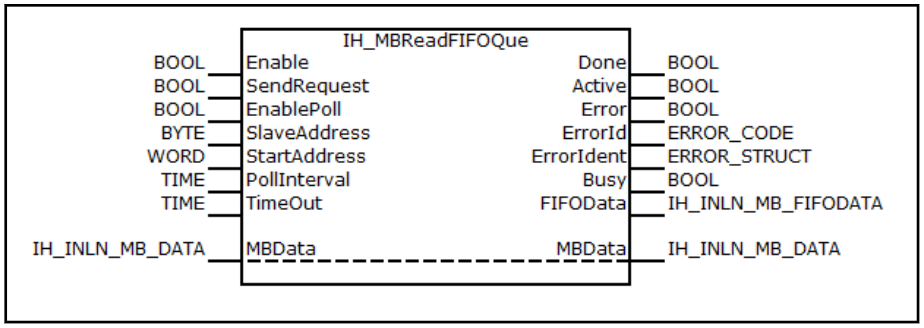


Fig. 20-20: IH_MBReadFIFOQue function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of Output Bits
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
	FIFOData	IH_INLN_MB_FIFODATA	Structure gives additional information about Byte count, FIFO count and FIFO register values IH_INLN_MB_FIFODATA, page 865
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-27: Interface variables of the IH_MBReadFIFOQue function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable

Name	Type	Min. value	Max. value	Default value	Effective
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	65535	0	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-28: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadFIFOQue

Functional description

The function block **IH_MBReadFIFOQue** (FC 24) is used to read First-In-First-Out, i.e. the FIFO queue of the register of the slave module. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program; its structure communicates and exchanges the data.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The function block **IH_MBReadFIFOQue** should be instantiated along with the IH_MBRTUMaster to communicate with the Modbus slave. Use a common variable at the MBData of **IH_MBReadFIFOQue** and IH_MBRTUMaster function blocks to exchange data with the Modbus slave.

In this example, the function block **IH_MBReadFIFOQue** is used to read the FIFO queue of the slave with address 1 and StartAddress 2550.

Enable both the IH_MBRTUMaster and **IH_MBReadFIFOQue** function blocks, with the rising edge at the bSendRequest input, the function block reads the FIFO queue of slave address 1 and the result is available in the stFIFOData.

By setting the bEnablePoll variable to TRUE and the Pollinterval to more than 1 second, the data is read from the slave in a regular interval.

After each successful completion of the read operation, the done bit is set to true for one PLC cycle.

Variable declaration

```
VAR
    fbIH_MBReadFIFOQue: IH_MBReadFIFOQue;
    bEnable: BOOL;
    bSendRequest: BOOL;
    bEnablePoll: BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    stFIFOData: IH_INLN_MB_FIFODATA;
END_VAR
```

Implementation

```
fbIH_MBReadFIFOQue(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= bEnablePoll,
    SlaveAddress:= BYTE#1,
    StartAddress:= WORD#2550,
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    FIFOData=> stFIFOData);
```

Response

Expression	Type	Value
stFIFOData	IH_INLN_MB_FIFODATA	
ByteCount	UINT	6
FIFOCount	UINT	2
FIFORegValues	ARRAY [1..31] OF WORD	
FIFORegValues[1]	WORD	100
FIFORegValues[2]	WORD	200

Fig. 20-21: stFIFOData

20.11 IH_MBReadFileRecord

Function code FC 20

Brief description The function block **IH_MBReadFileRecord** is used to perform a file record read in RTU mode. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-29: Reference table

Interface description

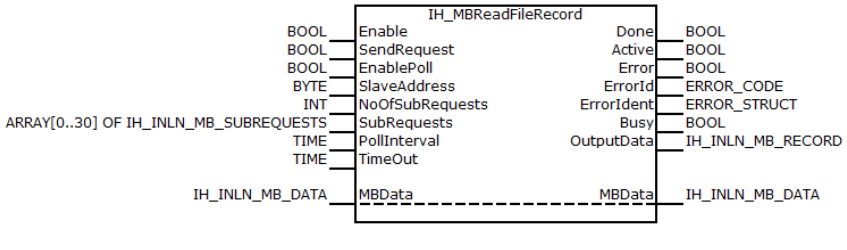


Fig. 20-22: IH_MBReadFileRecord function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	NoOfSubRequests	INT	Number of sub requests
	SubRequests	ARRAY[1..30] OF IH_INLN_MB_SUBREQUESTS	List of sub requests. Please refer to: Structures and enumerations
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with the master
	OutputData	IH_INLN_MB_RECORD	Contains request data length and sub requested data. Please refer to: Structures and enumerations IH_INLN_MB_RECORD, page 866
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-30: Interface variables of the IH_MBReadFileRecord function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
NoOfSubRequests	INT	1	30	1	Continuous with Enable

Name	Type	Min. value	Max. value	Default value	Effective
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-31: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadFileRecord

Functional description

The function block **IH_MBReadFileRecord** (FC 20) is used to perform a file record read in RTU mode. All Request Data Lengths in the structure (IH_INLN_MB_SUBREQUESTS) are provided in terms of number of bytes and all Record Lengths are provided in terms of registers. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program; its structure communicates and exchanges the data.

The function can read multiple groups of references. The groups can be separating (non-contiguous),

but the references within each group must be sequential.

- Each group is defined in a separate ‘sub-request’ field that contains 7 bytes:
- The reference type: 1 byte (must be specified as 6). Note: Considered default in this FB.
- The file number: (Word) should be (0x0001 to 0xFFFF).
- The Start record number within the file: (Word) should be (0x0000 to 0x270F).
- The length of the record to be read: (Word).

The quantity of registers to be read, combined with all other fields in the expected response,

must not exceed the valid length of the MODBUS PDU : 253 bytes.

As a result, Total record length of all SubRequests should be less then (123 – NoOfSubRequests).

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the IH_MBReadFileRecord function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBReadFileRecord function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBReadFileRecord function block is used to read the file records from the slave device.

Reading 2 file records with SendRequest condition

The **IH_MBReadFileRecord** used to read 2 Records of Record length 4 and 3 from the file num 1 and 2 respectively from slave Address 2 using "bSendRequest" as BOOL to trigger the Read file.

Variable declaration

```
VAR
    fbIH_MBReadFileRecord: IH_MBReadFileRecord;
    bEnable: BOOL;
    bSendRequest: BOOL;
    stSubRequests: ARRAY [1..30] OF IH_INLN_MB_SUBREQUESTS;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    stOutputData: IH_INLN_MB_RECORD;
END_VAR
```

Implementation

```
stSubRequests[1].FileNumber := WORD#1;
stSubRequests[1].RecordNumber := WORD#0;
stSubRequests[1].RecordLength := WORD#4;
stSubRequests[2].FileNumber := WORD#2;
stSubRequests[2].RecordNumber := WORD#5;
stSubRequests[2].RecordLength := WORD#3;


fbIH_MBReadFileRecord(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    NoOfSubRequests:= INT#2,
    SubRequests:= stSubRequests,
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    OutputData=> stOutputData);
```

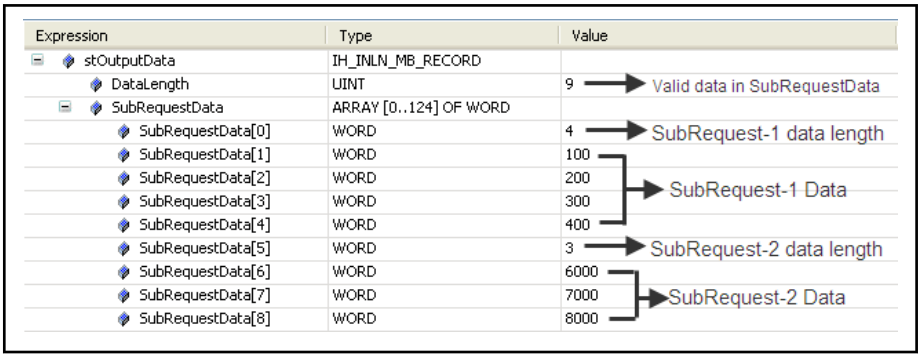


Fig. 20-23: stOutputData for program 1

Reading 1 file record with Poll Interval condition

The **IH_MBReadFileRecord** used to read 1 Record of Record length 6 from file num 1 from slave Address 5 using Poll interval (3 seconds) to trigger the read file.

Variable declaration

```
VAR
    fbIH_MBReadFileRecord2: IH_MBReadFileRecord;
    bEnable2: BOOL;
    stSubRequests2: ARRAY [1..30] OF IH_INLN_MB_SUBREQUESTS;
    stMBData: IH_INLN_MB_DATA;
    bDone2: BOOL;
```

```
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
    stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
    stOutputdata2: IH_INLN_MB_RECORD;
END_VAR
```

Implementation

```
stSubRequests2[1].FileNumber := WORD#1;
stSubRequests2[1].RecordNumber := WORD#0;
stSubRequests2[1].RecordLength := WORD#6;


fbIH_MBReadFileRecord2(
    Enable:= bEnable2,
    SendRequest:= FALSE,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#5,
    NoOfSubRequests:= INT#1,
    SubRequests:= stSubRequests2,
    PollInterval:= T#3S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
    ErrorID=> enErrorID2,
    ErrorIdent=> stErrorIdent2,
    Busy=> bBusy2,
    OutputData=> stOutputdata2);
```

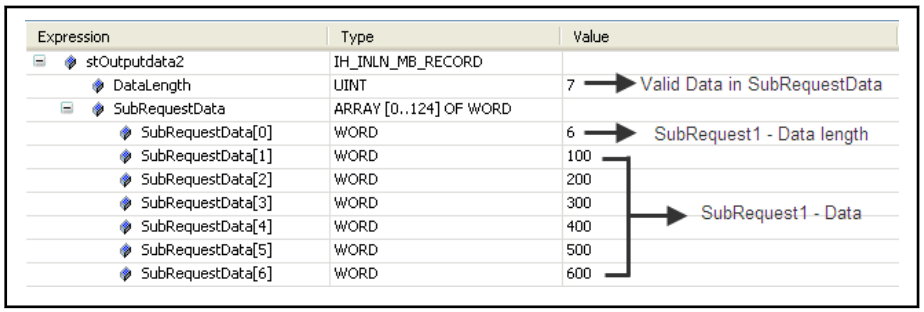


Fig. 20-24: stOutputData2 for program 2

20.12 IH_MBReadHoldingRegisters

Function code FC 3

Brief description The function block **IH_MBReadHoldingRegisters** is used to read the holding registers of the Modbus slave module. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-32: Reference table

Interface description

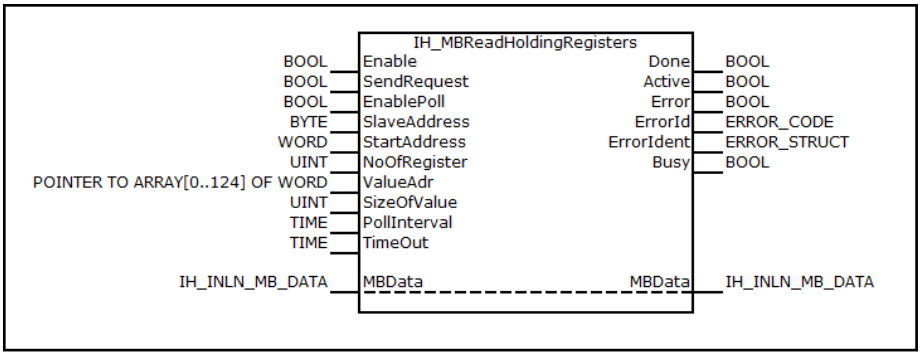


Fig. 20-25: IH_MBReadHoldingRegisters function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of holding register
	NoOfRegisters	UINT	Quantity of registers
	ValueAdr	POINTER TO ARRAY[0..124] OF WORD	Address of the Array to store Holding registers. Use ADR () operator
	SizeOfValue	UINT	Size of the Value array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_OUTPUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-33: Interface variables of the IH_MBReadHoldingRegisters function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	65535	0	Rising edge at Enable
NoOfRegisters	UINT	1	125	1	Rising edge at Enable
SizeOfValue	UINT	1	----------	--------	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-34: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadHoldingRegisters

Functional description

The function block **IH_MBReadHoldingRegisters** (FC 3) is used to read the holding registers of a the slave module specified at the SlaveAddress input. The function block should be called with the Modbus master function block. "MBData" should link to the master function block to communicate and exchanges the data with the master.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The function block IH_MBReadHoldingRegisters should be instantiated along with the IH_MBRTUMaster, page 835, to communicate with the Modbus slave. Use a common variable at the MBData of IH_MBReadHoldingRegisters and IH_MBRTUMaster function blocks to exchange the data with the Modbus slave.

Example 1

In this example, the function block IH_MBReadHoldingRegisters is used to read the holding register of slave address 1 with StartAddress 5 and NoOfRegisters 5.

Enable both IH_MBRTUMaster and IH_MBReadHoldingRegisters function blocks, with the rising edge at the bSendRequest input, the function block reads the data from slave address 1 and after successful completion of the read operation, the done bit is set to true for one PLC cycle.

Variable declaration

```
VAR
    fbIH_MBReadHoldingRegisters: IH_MBReadHoldingRegisters;
    bEnable: BOOL;
    bSendRequest: BOOL;
    bEnablePoll: BOOL;
    arHoldingRegister: ARRAY [0..4] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
```

```
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadHoldingRegisters(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= bEnablePoll,
    SlaveAddress:= BYTE#1,
    StartAddress:= WORD#5,
    NoOfRegisters:= UINT#5,
    ValueAdr:= ADR(arHoldingRegister),
    SizeOfValue:= SIZEOF(arHoldingRegister),
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

Response

The read data is available in the arHoldingRegister array.

Expression	Type	Value	
arHoldingRegister	ARRAY [0..4] OF WO...		
arHoldingRegister[0]	WORD	100	Data value of register 5
arHoldingRegister[1]	WORD	200	
arHoldingRegister[2]	WORD	300	
arHoldingRegister[3]	WORD	400	
arHoldingRegister[4]	WORD	500	Data value of register 9

Fig. 20-26: arHoldingRegister array

Example 2

In this example, the function block IH_MBReadHoldingRegisters is used to read the holding register of slave address 2 with StartAddress 100 and NoOfRegisters 10 with a poll interval of 1 second.

Enable both IH_MBRTUMaster and IH_MBReadHoldingRegisters function blocks and set EnablePoll to true with a PollInterval of 1 second. Subsequently, the function block reads the data from slave address 2 with a regular interval of 1 second and after each successful completion of the read operation, the done bit is set to true for one PLC cycle.

Variable declaration

```
VAR
    fbIH_MBReadHoldingRegisters_2: IH_MBReadHoldingRegisters;
    bEnable2: BOOL;
    bSendRequest2: BOOL;
    arHoldingRegister2: ARRAY [0..9] OF WORD;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
    stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
stMBData: IH_INLN_MB_DATA;

END_VAR
```

Implementation

```
fbIH_MBReadHoldingRegisters_2(
    Enable:= bEnable2,
    SendRequest:= bSendRequest2,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#100,
    NoOfRegisters:= UINT#10,
    ValueAdr:= ADR(arHoldingRegister2),
    SizeOfValue:= SIZEOF(arHoldingRegister2),
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
    ErrorID=> enErrorID2,
    ErrorIdent=> stErrorIdent2,
    Busy=> bBusy2);
```

Response

The read data is available in the arHoldingRegister2 array.

Expression	Type	Value	
arHoldingRegister2	ARRAY [0..9] OF WORD		
arHoldingRegister2[0]	WORD	600	Data value of register 100
arHoldingRegister2[1]	WORD	700	Data value of register 101
arHoldingRegister2[2]	WORD	800	
arHoldingRegister2[3]	WORD	900	
arHoldingRegister2[4]	WORD	1000	
arHoldingRegister2[5]	WORD	1100	
arHoldingRegister2[6]	WORD	1200	
arHoldingRegister2[7]	WORD	1300	
arHoldingRegister2[8]	WORD	1400	Data value of register 108
arHoldingRegister2[9]	WORD	1500	Data value of register 109

Fig. 20-27: arHoldingRegister2 array

20.13 IH_MBReadInputRegisters

Function code FC 4

Brief description The function block **IH_MBReadInputRegisters** is used to read the input registers in RTU mode. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-35: Reference table

Interface description

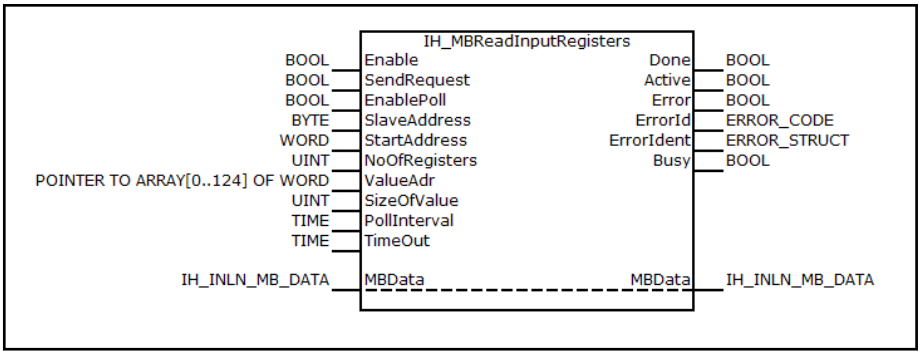


Fig. 20-28: IH_MBReadInputRegisters function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of registers
	NoOfRegisters	UINT	Quantity of registers to read
	ValueAdr	POINTER TO ARRAY[0..124] OF WORD	Address of the Array to store Input registers. Use ADR () operator
	SizeOfValue	UINT	Size of the Value array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-36: Interface variables of the IH_MBReadInputRegisters function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	16#FFFF	0	Rising edge at Enable
NoOfRegisters	UINT	1	125	1	Rising edge at Enable
SizeOfValue	UINT	1	----------	---------	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-37: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadInputRegisters

Functional description

The function block **IH_MBReadInputRegisters** (FC 4) is used to read the input registers in RTU mode. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program; its structure communicates and exchanges the data.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the the IH_MBReadInputRegisters function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between thethe IH_MBRTUMaster and IH_MBReadInputRegisters function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBReadInputRegisters function block is used to read the input Registers from the slave device.

Reading input registers with SendRequest condition

The IH_MBReadInputRegisters used to read 10 input registers of the Start address 0x1000 from slave Address 2 using "bSendRequest" as BOOL to trigger the read registers.

Variable declaration

```
VAR
    fbIH_MBReadInputRegisters: IH_MBReadInputRegisters;
    bEnable: BOOL;
    bSendrequest: BOOL;
    arInputRegisters: ARRAY [0..124] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadInputRegisters(
    Enable:= bEnable,
    SendRequest:= bSendrequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#1000,
    ValueAdr:= ADR(arInputRegisters),
    NoOfRegisters:= UINT#10,
    SizeOfValue:= SIZEOF(arInputRegisters),
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

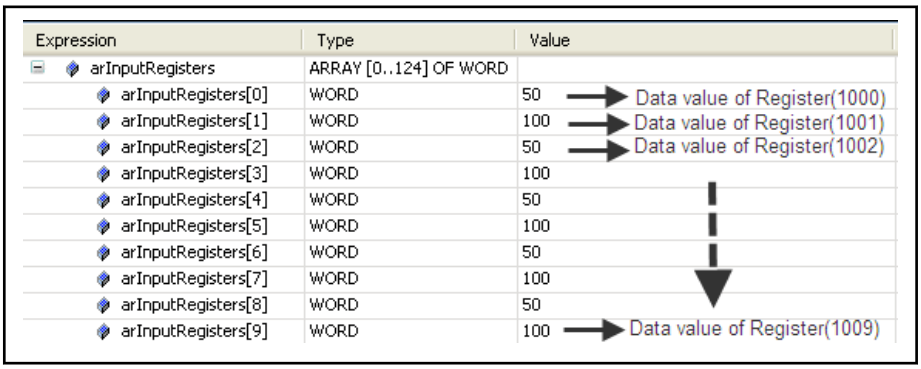


Fig. 20-29: arInputRegisters for program 1

Reading input registers with Poll interval condition

The IH_MBReadInputRegisters is used to read 5 input registers of the Start address 0x0500 from slave Address 5 using "Poll Interval" (3 Seconds) to trigger the read registers.

Variable declaration

```
VAR
    fbIH_MBReadInputRegisters2: IH_MBReadInputRegisters;
    bEnable2: BOOL;
    arInputRegisters2: ARRAY [0..7] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
      stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadInputRegisters2(
    Enable:= bEnable2,
    SendRequest:= FALSE,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#5,
    StartAddress:= WORD#500,
    ValueAdr:= ADR(arInputRegisters2),
    NoOfRegisters:= UINT#5,
    SizeOfValue:= SIZEOF(arInputRegisters2),
    PollInterval:= T#3S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
```

```
ErrorID=> enErrorID2,
ErrorIdent=> stErrorIdent2,
Busy=> bBusy2);
```

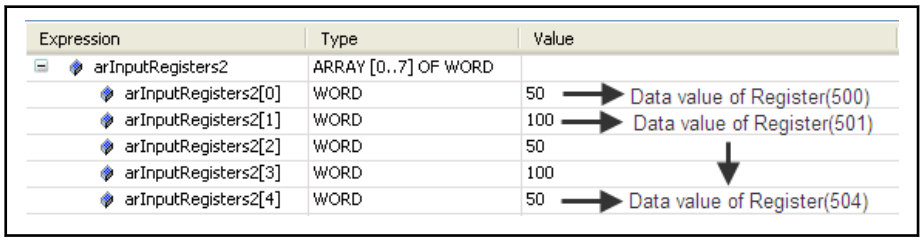


Fig. 20-30: arInputRegisters2 for program 2

20.14 IH_MBReadWriteMultipleRegisters

Function code FC 23

Brief description The function block **IH_MBReadWriteMultipleRegisters** performs a combination of one read and one write operation in a single Modbus transaction. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-38: Reference table

Interface description

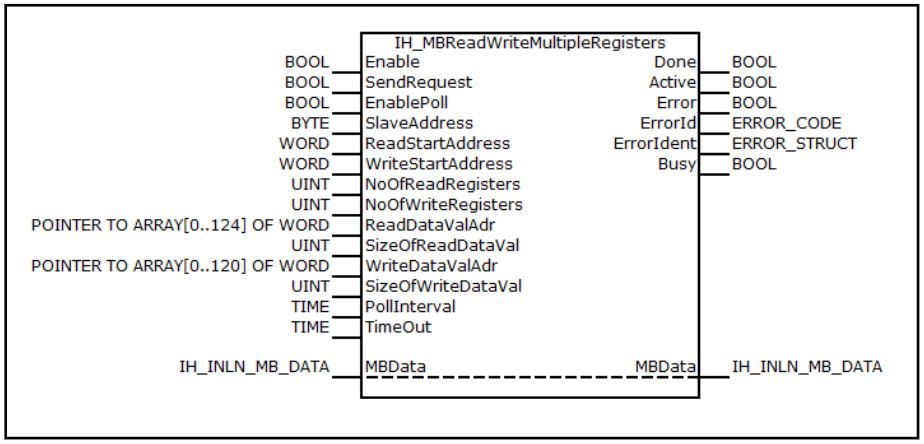


Fig. 20-31: IH_MBReadWriteMultipleRegisters function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	ReadStartAddress	WORD	Start address of holding register is applicable for Read operation
	WriteStartAddress	WORD	Start address of holding register is applicable for Write operation
	NoOfReadRegisters	UINT	Quantity of registers to read
	NoOfWriteRegisters	UINT	Quantity of registers to write
	ReadDataValAdr	POINTER TO ARRAY[0..124] OF WORD	Address of the Array to store read value of the registers. Use ADR () operator
	SizeOfReadDataVal	UINT	Size of the ReadDataVal array. Use SIZEOF () operator
	WriteDataValAdr	POINTER TO ARRAY[0..120] OF WORD	Address of the Array to contains write value of the registers. Use ADR () operator
	SizeOfWriteDataVal	UINT	Size of the WriteDataVal array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-39: Interface variables of the IH_MBReadWriteMultipleRegisters function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
ReadStartAddress	WORD	0	65535	0	Rising edge at Enable
WriteStartAddress	WORD	0	65535	0	Rising edge at Enable
NoOfReadRegisters	UINT	1	125	1	Rising edge at Enable
NoOfWriteRegisters	UINT	1	121	1	Rising edge at Enable
SizeOfReadDataVal	UINT	1	----------	----------	Rising edge at Enable
SizeOfWriteDataVal	UINT	1	----------	----------	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-40: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBReadWriteMultipleRegisters

Functional description

The function block **IH_MBReadWriteMultipleRegisters** (FC 23) performs a combination of one read and one write operation in a single Modbus transaction. The function block should be called with the Modbus master function block. "MBData" should link to the master function block to communicate and exchanges the data with the master.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block:

The IH_MBRTUMaster function block should be instantiated along with the **IH_MBReadWriteMultipleRegisters** function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and **IH_MBReadWriteMultipleRegisters** function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, IH_MBWriteMultipleCoils is used to read and write registers to the slave device.

Read and write registers with a SendRequest condition

The examples shows writing and reading data of 5 registers of the start address 0x4000 of a Modbus Slave device with address 1. A rising edge at the "bSendRequest" input, triggers the read & write operation.

Variable declaration

```
VAR
    fbIH_MBReadWriteMultipleRegisters: IH_MBReadWriteMultipleRegisters;
    bEnable: BOOL;
    bSendRequest: BOOL;
    arReadData: ARRAY [0..4] OF WORD;
    arWriteData: ARRAY [0..4] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadWriteMultipleRegisters(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#1,
    ReadStartAddress:= WORD#4000,
    WriteStartAddress:= WORD#4000,
    NoOfReadRegisters:= UINT#5,
    NoOfWriteRegisters:= UINT#5,
    ReadDataValAdr:= ADR(arReadData),
    SizeOfReadDataVal:= SIZEOF(arReadData),
    WriteDataValAdr:= ADR(arWriteData),
    SizeOfWriteDataVal:= SIZEOF(arWriteData),
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

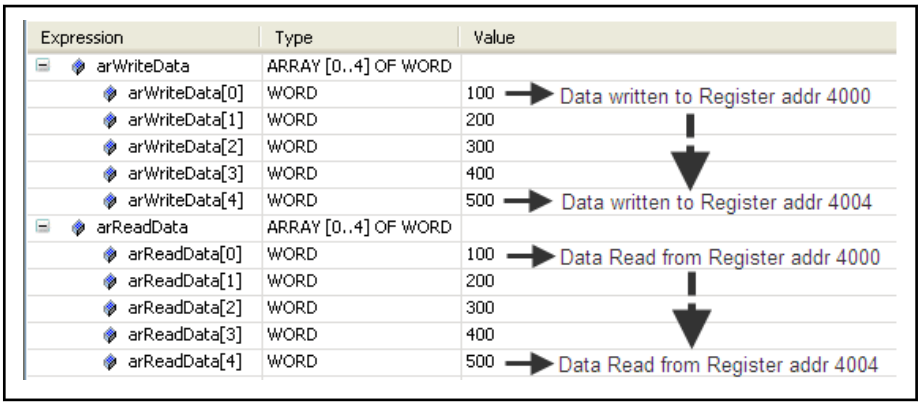


Fig. 20-32: Read and write data to slave device in program 1

Read and write registers with SendRequest condition

This example shows writing of 10 registers of the Start address 0x4200 and reading of 5 registers from the Start address 0x4100 of a Modbus RTU Slave with the address 1.

The "PollInterval" is configured to 3 seconds and "EnablePoll" is set to TRUE.

A rising edge at the input "bSendRequest", triggers the read & write operation.

Variable declaration

```
VAR
    fbIH_MBReadWriteMultipleRegisters2: IH_MBReadWriteMultipleRegisters;
    bEnable2: BOOL;
    arReadData2: ARRAY [0..4] OF WORD;
    arWriteData2: ARRAY [0..9] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
    stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
END_VAR
```

Implementation

```
fbIH_MBReadWriteMultipleRegisters2(
    Enable:= bEnable2,
    SendRequest:= FALSE,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#1,
    ReadStartAddress:= WORD#4100,
    WriteStartAddress:= WORD#4200,
    NoOfReadRegisters:= UINT#5,
    NoOfWriteRegisters:= UINT#10,
    ReadDataValAdr:= ADR(arReadData2),
    SizeOfReadDataVal:= SIZEOF(arReadData2),
    WriteDataValAdr:= ADR(arWriteData2),
    SizeOfWriteDataVal:= SIZEOF(arWriteData2),
    PollInterval:= T#3S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
    ErrorID=> enErrorID2,
    ErrorIdent=> stErrorIdent2,
    Busy=> bBusy2);
```

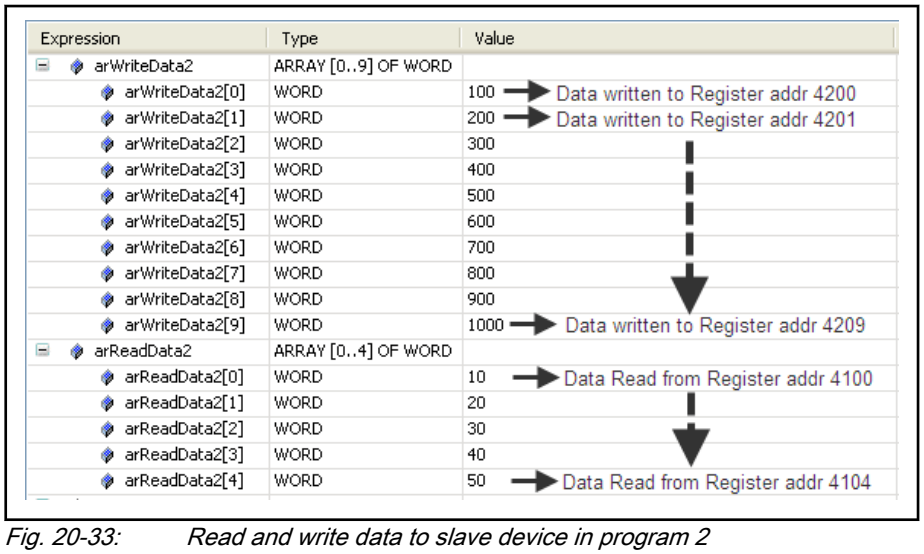


Fig. 20-33: Read and write data to slave device in program 2

20.15 IH_MBRTUMaster

Brief description

The function block **IH_MBRTUMaster** ensures communication between the controller and the physical serial network (slaves) in RTU mode. It receives a request from other function block function codes and sends it to slaves through the **R-IB IL RS 485_422 PRO/R-IB IL RS232 Pro/R-IB IL RS UNI** modules. It receives a response from a slave and sends it back to the block that initially sent the request. For each terminal, the function block has to be

instantiated once. It communicates with a maximum of 33 function blocks of function codes at a time.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-41: Reference table

Interface description

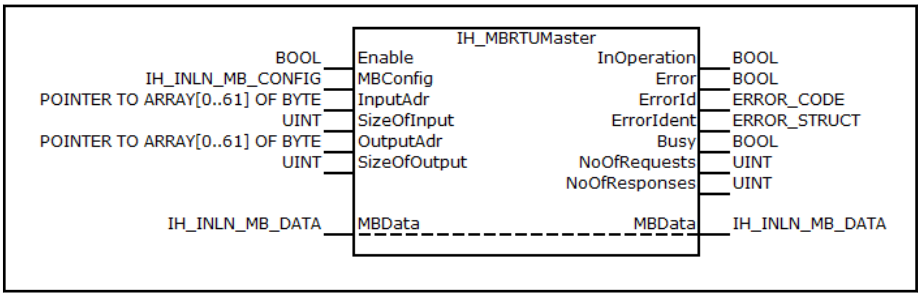


Fig. 20-34: IH_MBRTUMaster function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	MBConfig	IH_INLN_MB_CONFIG	Used to configure Modbus. IH_INLN_MB_CONFIG, page 865
	InputAdr	POINTER TO ARRAY[0..61] OF BYTE	Address of input process data array. Use ADR () operator
	SizeOfInput	UINT	Size of the InputAdr array. Use SIZEOF () operator
	OutputAdr	POINTER TO ARRAY[0..61] OF BYTE	Address of output process data array. Use ADR () operator
	SizeOfOutput	UINT	Size of the OutputAdr array. Use SIZEOF () operator
VAR_OUTPUT	InOperation	BOOL	Function block is active on its actual task
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with function blocks of function code
	NoOfRequests	UINT	Number of sent requests
	NoOfResponses	UINT	Number of received responses
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w IH_MBRTUMaster and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-42: Interface variables of the IH_MBRTUMaster function block

Min./Max. and Default Values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SizeOfInput	UINT	12	62	----------	Rising edge at Enable
SizeOfOutput	UINT	12	62	----------	Rising edge at Enable

Tab. 20-43: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBRTUMaster

Functional description

The function block **IH_MBRTUMaster** ensures communication between the controller and physical serial network (slaves) in RTU mode. It receives requests from other function codes function blocks and sends it to slaves through the **R-IB IL RS 485_422 PRO/R-IB IL RS232 PRO/R-IB IL RS UNI** modules. It receives a response from a slave, sends it back to the function block that sent the request. The function block ensures communication with the terminal. The "InputAdr" and "OutputAdr" should be mapped to the terminal. "MBData" should link to other function blocks of function codes in the program, its structure communicates with the other blocks and exchanges the data. It communicates with maximum of 33 function blocks of function codes

at a time.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "InOperation" output is set to TRUE. If queued function codes are detected, the function block will start processing until the "Enable" bit is TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Handling

1. If the 7 bits data width is selected for communication, the MODBUS master or slave device may generate a CRC error. Preferred data width for communication is 8 bits
2. While using master and slave function blocks in MLD, data bytes are swapped and the data received is also swapped

Example of a function block:

The IH_MBRTUMaster function block should be instantiated with other Modbus function blocks to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and other function code function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBRTUMaster function block is configured to exchange data with the Modbus unit and other Modbus function blocks. "Input ProcessData" and "Output ProcessData" variables should be assigned to the Dali unit for communication.

Communicating with the RS232Pro module

The **IH_MBRTUMaster** used to communicate with the Modbus slave devices through RS232Pro module.

Variable declaration

```
VAR
    fbIH_MBRTUMaster: IH_MBRTUMaster;
 bEnable: BOOL;
    stMBConfig: IH_INLN_MB_CONFIG;
    arOutprocData_RS232: ARRAY [0..11] OF BYTE;
    arInprocData_RS232: ARRAY [0..11] OF BYTE;
    stMBData: IH_INLN_MB_DATA;
```

```
    bInOperation: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    uiNoOfRequests: UINT;
    uiNoOfResponses: UINT;

END_VAR
```

Implementation

```
stMBConfig.Baudrate := IL_INLN_RS_BAUDRATE.BAUDRATE_19200;
stMBConfig.DataWidth := IL_INLN_RS_DATAWIDTH.DATABIT8_EVEN_STOPBIT2;
stMBConfig.InterfaceType := IL_INLN_RS_INTERFACE.RS232;
stMBConfig.ModuleType :=  IL_INLN_RS_MODULETYPE.RS232PRO;


fbIH_MBRTUMaster(
    Enable:= bEnable,
    MBConfig:= stMBConfig,
    InputAdr:= ADR(arInprocData_RS232),
    SizeOfInput:= SIZEOF(arInprocData_RS232),
    OutputAdr:= ADR(arOutprocData_RS232),
    SizeOfOutput:= SIZEOF(arOutprocData_RS232),
    MBData:= stMBData,
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    NoOfRequests=> uiNoOfRequests,
    NoOfResponses=> uiNoOfResponses);
```

Process data mapping

To create the process data mapping, two alternatives are possible. It may depends on the given mapping of the used module. The suggestion will be **alternative I**, because its easy and compact.

Alternative I

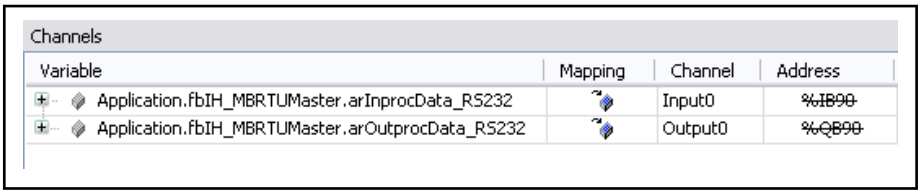


Fig. 20-35: Process data mapping IH_MBRTUMaster

Alternative II

Channels

Variable	Mapping	Channel	Address	Type
		Input0	%IB1	
Application.fbIH_MBRTUMaster.arInprocData_RS232[0]		Byte0	%IB1	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[1]		Byte1	%IB2	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[2]		Byte2	%IB3	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[3]		Byte3	%IB4	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[4]		Byte4	%IB5	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[5]		Byte5	%IB6	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[6]		Byte6	%IB7	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[7]		Byte7	%IB8	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[8]		Byte8	%IB9	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[9]		Byte9	%IB10	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[10]		Byte10	%IB11	BYTE
Application.fbIH_MBRTUMaster.arInprocData_RS232[11]		Byte11	%IB12	BYTE
		Output0	%QB1	
Application.fbIH_MBRTUMaster.arOutprocData_RS232[0]		Byte0	%QB1	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[1]		Byte1	%QB2	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[2]		Byte2	%QB3	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[3]		Byte3	%QB4	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[4]		Byte4	%QB5	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[5]		Byte5	%QB6	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[6]		Byte6	%QB7	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[7]		Byte7	%QB8	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[8]		Byte8	%QB9	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[9]		Byte9	%QB10	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[10]		Byte10	%QB11	BYTE
Application.fbIH_MBRTUMaster.arOutprocData_RS232[11]		Byte11	%QB12	BYTE

Fig. 20-36: Process data mapping IH_MBRTUMaster

Communicating with the RSUni (RS485 with 15 words) module

The **IH_MBRTUMaster** used to communicate with the Modbus slave devices through RSUni with 15 words module. With BaudRate: 19200, DataWidth: DATABIT8_EVEN_STOPBIT1.

Variable declaration

```
VAR
 fbIH_MBRTUMasterRSUni: IH_MBRTUMaster;
    bEnableRSUni: BOOL;
    stMBConfigRSUni: IH_INLN_MB_CONFIG;
 arInProcData_RSUNI_15WORDS: ARRAY [0..29] OF BYTE;
    arOutprocData_RSUNI_15WORDS: ARRAY [0..29] OF BYTE;
    bInOperationRSUni: BOOL;
    bErrorRSUni: BOOL;
    enErrorIDRSUni: ERROR_CODE;
    stErrorIdentRSUni: ERROR_STRUCT;
    bBusyRSUni: BOOL;
    uiNoOfRequestsRSUni: UINT;
    uiNoOfResponsesRSUni: UINT;
END_VAR
```

Implementation

```
stMBConfigRSUni.Baudrate := IL_INLN_RS_BAUDRATE.BAUDRATE_19200;
stMBConfigRSUni.DataWidth := IL_INLN_RS_DATAWIDTH.DATABIT8_EVEN_STOPBIT1;
stMBConfigRSUni.InterfaceType := IL_INLN_RS_INTERFACE.RS485;
stMBConfigRSUni.ModuleType :=  IL_INLN_RS_MODULETYPE.RSUNI_15WORDS;


fbIH_MBRTUMasterRSUni(
    Enable:= bEnableRSUni,
    MBConfig:= stMBConfigRSUni,
    InputAdr:= ADR(arInProcData_RSUNI_15WORDS),
    SizeOfInput:= SIZEOF(arInProcData_RSUNI_15WORDS),
    OutputAdr:= ADR(arOutprocData_RSUNI_15WORDS),
    SizeOfOutput:= SIZEOF(arOutprocData_RSUNI_15WORDS),
    MBData:= stMBData,
    InOperation=> bInOperationRSUni,
    Error=> bErrorRSUni,
```

```
ErrorID=> enErrorIDRSUni,
ErrorIdent=> stErrorIdentRSUni,
Busy=> bBusyRSUni,
NoOfRequests=> uiNoOfRequestsRSUni,
NoOfResponses=> uiNoOfResponsesRSUni);
```

20.16 IH_MBRTUSlave

Function code FC 1, 2, 3, 4, 5, 6, 15, 16, 22 and 23

Brief description The function block **IH_MBRTUSlave** ensures communication between the controller and physical serial network (master) in RTU mode. It enables the controller (**R-IB IL RS 485_422 PRO / R-IB IL RS232 Pro / R-IB IL RS UNI** modules) to act as a slave with the address specified at the SlaveAddress input. The supported function codes are: 1, 2, 3, 4, 5, 6, 15, 16, 22 and 23.

For each terminal, the function block has to be instantiated once.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-44: Reference table

Interface description

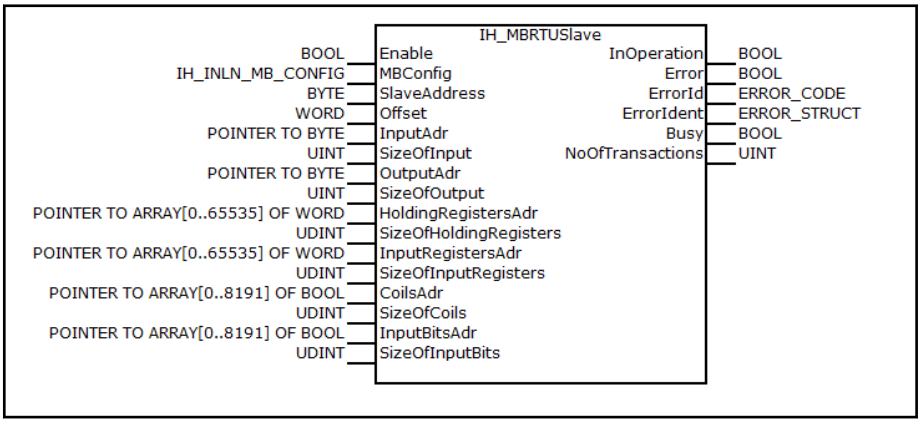


Fig. 20-37: IH_MBRTUSlave function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	MBConfig	IH_INLN_MB_CONFIG	Used to configure. IH_INLN_MB_CONFIG, page 865
	SlaveAddress	BYTE	Slave address
	Offset	WORD	Address offset. For example for Offset of 2000, for holding register address number 3 in HoldingRegisters array will be address 2003 (2000 + 3) instead of 3
	InputAdr	POINTER TO ARRAY[0..61] OF BYTE	Address of the Array to store input process data. Use ADR () operator
	SizeOfInput	UINT	Size of the InputAdr array. Use SIZEOF () operator
	OutputAdr	POINTER TO ARRAY[0..61] OF BYTE	Address of the Array to store output process data. Use ADR () operator
	SizeOfOutput	UINT	Size of the OutputAdr array. Use SIZEOF () operator
	HoldingRegistersAdr	POINTER TO ARRAY[0..65535] OF WORD	Address of the Array to store HoldingRegisters. Use ADR () operator
	SizeOfHolding Registers	UDINT	Size of the HoldingRegisters array. Use SIZEOF () operator
	InputRegistersAdr	POINTER TO ARRAY[0..65535] OF WORD	Address of the Array to store InputRegisters. Use ADR () operator
	SizeOfInputRegisters	UDINT	Size of the InputRegisters array. Use SIZEOF () operator
	CoilsAdr	POINTER TO ARRAY[0..8191] OF BOOL	Address of the Array to store Coils data. Use ADR () operator
	SizeOfCoils	UDINT	Size of the Coils array. Use SIZEOF () operator
	InputBitsAdr	POINTER TO ARRAY[0..8191] OF BOOL	Address of the Array to store InputBits. Use ADR () operator
	SizeOfInputBits	UDINT	Size of the InputBits array. Use SIZEOF () operator
VAR_OUTPUT	InOperation	BOOL	Function block is active on its actual task
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with the master
	NoOfTransactions	UINT	Number of transactions

Tab. 20-45: Interface variables of the IH_MBRTUSlave function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
Offset	WORD	0	65536	0	Rising edge at Enable
SizeOfInput	UINT	12	62	----------	Rising edge at Enable
SizeOfOutput	UINT	12	62	----------	Rising edge at Enable
SizeOfHolding-Registers	UDINT	1	65535	----------	Rising edge at Enable
SizeOfInputRegis-ters	UDINT	1	65535	----------	Rising edge at Enable
SizeOfCoils	UDINT	1	8192	----------	Rising edge at Enable
SizeOfInputBits	UDINT	1	8192	----------	Rising edge at Enable

Tab. 20-46: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBRTUSlave

Functional description

The function block **IH_MBRTUSlave** enables the controller to act as a slave in RTU mode. It enables the controller (**R-IB IL RS 485_422 PRO/R-IB IL RS232 Pro/R-IB IL RS UNI** modules) to act as a slave with the address specified at the SlaveAddress input. It has arrays of data emulating registers and bits as if it is real memory of a slave device.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation "InOperation" output is set to TRUE. If queued function codes are detected, the function block will start processing until the "Enable" bit is TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Handling

1. If a 7 bits data width is selected for communication, the MODBUS master or slave device can generate a CRC error. The preferred data width for communication is 8 bits
2. While using master and slave function blocks in MLD, data bytes are exchanged and the data received is also exchanged

Example of a function block

The IH_MBRTUSlave function block should be instantiated once for each Modbus module to acts as slave with the address specified at the SlaveAddress input.

Example 1

In this example, the function block is used to configure the R IB IL RS485 PRO module as Modbus slave.

Create an array of Holding Registers (arHoldingRegisters), Input Registers (arInputRegisters), Coils (arCoils) and Input Bits (arInputBits) before enabling the function block

Also create an array of Inprocess data (arInProcessData) and Outprocess data (arOutProcessData) and map the variables with process data of the R IB IL RS 485 PRO module

Assign the address at the SlaveAddress input and enable the function block. The function block sets the InOperation variable to TRUE to indicate the slave is ready to operate

Variable declaration

```
VAR
    fbIH_MBRTUSlave: IH_MBRTUSlave;
    bEnable: BOOL;
    arInProcessData: ARRAY [0..11] OF BYTE;
    arOutProcessData: ARRAY [0..11] OF BYTE;
    arHoldingRegisters: ARRAY [0..65535] OF WORD;
    arInputRegisters: ARRAY [0..65535] OF WORD;
    arCoils: ARRAY [0..8191] OF BOOL;
    arInputBits: ARRAY [0..8191] OF BOOL;
    bInOperation: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    uiNoOfTransactions: UINT;
END_VAR
```

Implementation

```
fbIH_MBRTUSlave.MBConfig.Baudrate := IL_INLN_RS_BAUDRATE.BAUDRATE_19200;
fbIH_MBRTUSlave.MBConfig.DataWidth := IL_INLN_RS_DATAWIDTH.DATABIT8_EVEN_STOPBIT2;
fbIH_MBRTUSlave.MBConfig.ModuleType := IL_INLN_RS_MODULETYPE.RS485PRO;
fbIH_MBRTUSlave.MBConfig.InterfaceType := IL_INLN_RS_INTERFACE.RS485;

fbIH_MBRTUSlave(
    Enable:= bEnable,
    SlaveAddress:= BYTE#1,
    Offset:= WORD#0,
    InputAdr:= ADR(arInProcessData),
    SizeOfInput:= SIZEOF(arInProcessData),
    OutputAdr:= ADR(arOutProcessData),
    SizeOfOutput:= SIZEOF(arOutProcessData),
    HoldingRegistersAdr:= ADR(arHoldingRegisters),
    SizeOfHoldingRegisters:= SIZEOF(arHoldingRegisters),
    InputRegistersAdr:= ADR(arInputRegisters),
    SizeOfInputRegisters:= SIZEOF(arInputRegisters),
    CoilsAdr:= ADR(arCoils),
    SizeOfCoils:= SIZEOF(arCoils),
    InputBitsAdr:= ADR(arInputBits),
    SizeOfInputBits:= SIZEOF(arInputBits),
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    NoOfTransactions=> uiNoOfTransactions);
```

Process data mapping

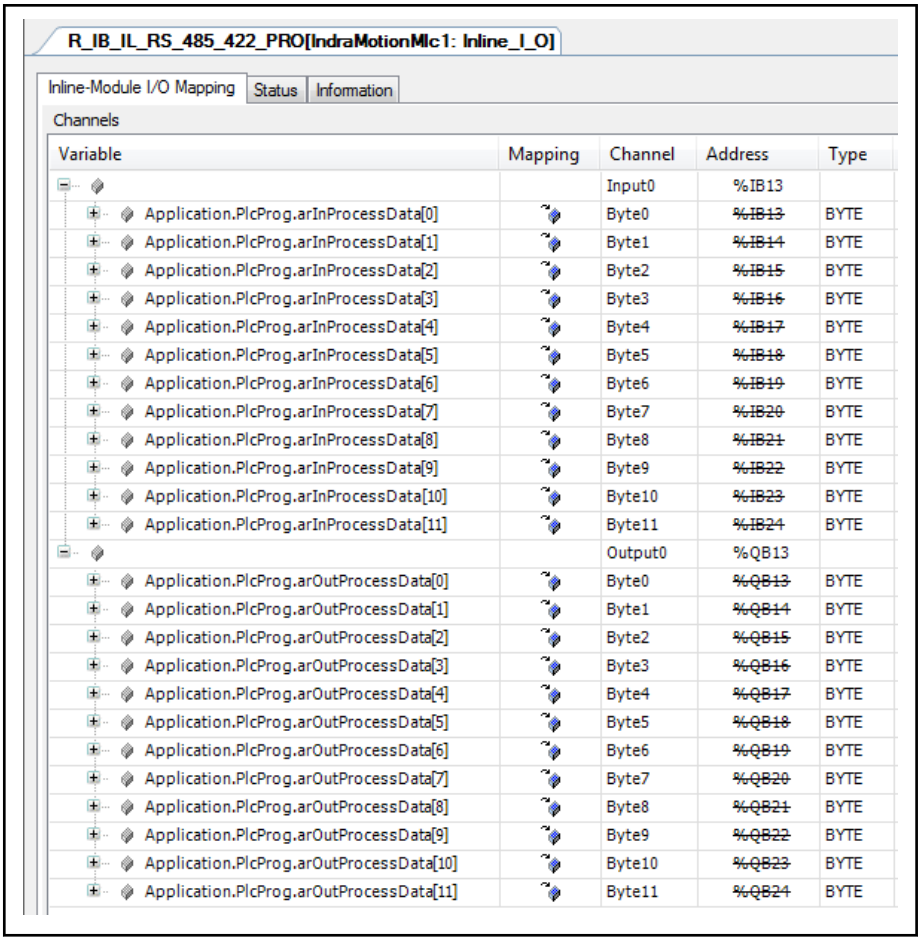


Fig. 20-38: Process data mapping IH_MBRTUSlave

Example 2

In this example, the function block is used to configure the R IB IL RSUNI module as Modbus slave.

Create an array of Holding Registers (arHoldingRegistersUni), Input Registers (arInputRegistersUni), Coils (arCoilsUni) and Input Bits (arInputBitsUni) before enabling the function block

Also create an array of Inprocess data (arInProcessDataUni) and Outprocess data (arOutProcessDataUni) and map the variables with the process data of the R IB IL RS UNI module

Assign the address at the SlaveAddress input and enable the function block. The function block sets the InOperation variable to TRUE to indicate the slave is ready for operation

Variable declaration

```
VAR
    fbIH_MBRTUSlaveUni: IH_MBRTUSlave;
    bEnableUni: BOOL;
    arInProcessDataUni: ARRAY [0..13] OF BYTE;
    arOutProcessDataUni: ARRAY [0..13] OF BYTE;
    arHoldingRegistersUni: ARRAY [0..65535] OF WORD;
    arInputRegistersUni: ARRAY [0..65535] OF WORD;
    arCoilsUni: ARRAY [0..8191] OF BOOL;
    arInputBitsUni: ARRAY [0..8191] OF BOOL;
    bInOperationUni: BOOL;
    bErrorUni: BOOL;
    enErrorIDUni: ERROR_CODE;
```

```
    stErrorIdentUni: ERROR_STRUCT;
    bBusyUni: BOOL;
    uiNoOfTransactionsUni: UINT;
END_VAR
```

Implementation

```
fbIH_MBRTUSlaveUni.MBConfig.Baudrate := IL_INLN_RS_BAUDRATE.BAUDRATE_19200;
fbIH_MBRTUSlaveUni.MBConfig.DataWidth := IL_INLN_RS_DATAWIDTH.DATABIT8_EVEN_STOPBIT2;
fbIH_MBRTUSlaveUni.MBConfig.ModuleType := IL_INLN_RS_MODULETYPE.RSUNI_7WORDS;
fbIH_MBRTUSlaveUni.MBConfig.InterfaceType := IL_INLN_RS_INTERFACE.RS485;

fbIH_MBRTUSlaveUni(
    Enable:= bEnableUni,
    SlaveAddress:= BYTE#1,
    Offset:= WORD#0,
    InputAdr:= ADR(arInProcessDataUni),
    SizeOfInput:= SIZEOF(arInProcessDataUni),
    OutputAdr:= ADR(arOutProcessDataUni),
    SizeOfOutput:= SIZEOF(arOutProcessDatauni),
    HoldingRegistersAdr:= ADR(arHoldingRegistersUni),
    SizeOfHoldingRegisters:= SIZEOF(arHoldingRegistersUni),
    InputRegistersAdr:= ADR(arInputRegistersUni),
    SizeOfInputRegisters:= SIZEOF(arInputRegistersUni),
    CoilsAdr:= ADR(arCoilsUni),
    SizeOfCoils:= SIZEOF(arCoilsUni),
    InputBitsAdr:= ADR(arInputBitsUni),
    SizeOfInputBits:= SIZEOF(arInputBitsuni),
    InOperation=> bInOperationUni,
    Error=> bErrorUni,
    ErrorID=> enErrorIDUni,
    ErrorIdent=> stErrorIdentUni,
    Busy=> bBusyUni,
    NoOfTransactions=> uiNoOfTransactionsUni);
```

Mapping process data of RS UNI on local (Inline) bus

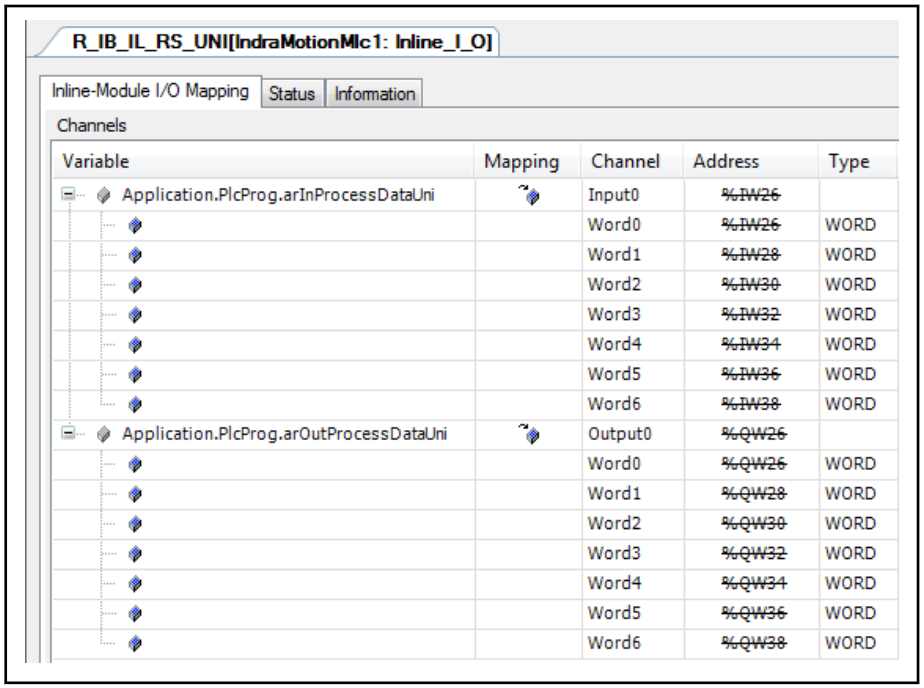


Fig. 20-39: Mapping process data of RS UNI on local (Inline) bus

Mapping process data of RS UNI on sercos bus

IB_IL_RS_UNI_PD_112_11_Byte_user_data_[IndraMotionMlc1: Sercos_III_Master: R_IL_S3_BK_DI8_DO4_PAC]

sercos Module | Function Groups | User Parameter | sercos Module I/O Mapping | Information

Channels

Variable	Mapping	Channel	Address	Type
		DiagOut	%QB41	
Application.prgIH_MBRTUSlave.arOutProcessDataUni[0]		Byte 0	~~%QB41~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[1]		Byte 1	~~%QB42~~	BYTE
		Complex Protocol	%QB43	
Application.prgIH_MBRTUSlave.arOutProcessDataUni[2]		IL RS UNI - No of Bytes	~~%QB43~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[3]		IL RS UNI - Byte out 0	~~%QB44~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[4]		IL RS UNI - Byte out 1	~~%QB45~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[5]		IL RS UNI - Byte out 2	~~%QB46~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[6]		IL RS UNI - Byte out 3	~~%QB47~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[7]		IL RS UNI - Byte out 4	~~%QB48~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[8]		IL RS UNI - Byte out 5	~~%QB49~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[9]		IL RS UNI - Byte out 6	~~%QB50~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[10]		IL RS UNI - Byte out 7	~~%QB51~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[11]		IL RS UNI - Byte out 8	~~%QB52~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[12]		IL RS UNI - Byte out 9	~~%QB53~~	BYTE
Application.prgIH_MBRTUSlave.arOutProcessDataUni[13]		IL RS UNI - Byte out 10	~~%QB54~~	BYTE
		DiagIn	%IB42	
Application.prgIH_MBRTUSlave.arInProcessDataUni[0]		Byte 0	~~%IB42~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[1]		Byte 1	~~%IB43~~	BYTE
		Complex Protocol	%IB44	
Application.prgIH_MBRTUSlave.arInProcessDataUni[2]		IL RS UNI - No of Bytes	~~%IB44~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[3]		IL RS UNI - Byte in 0	~~%IB45~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[4]		IL RS UNI - Byte in 1	~~%IB46~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[5]		IL RS UNI - Byte in 2	~~%IB47~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[6]		IL RS UNI - Byte in 3	~~%IB48~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[7]		IL RS UNI - Byte in 4	~~%IB49~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[8]		IL RS UNI - Byte in 5	~~%IB50~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[9]		IL RS UNI - Byte in 6	~~%IB51~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[10]		IL RS UNI - Byte in 7	~~%IB52~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[11]		IL RS UNI - Byte in 8	~~%IB53~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[12]		IL RS UNI - Byte in 9	~~%IB54~~	BYTE
Application.prgIH_MBRTUSlave.arInProcessDataUni[13]		IL RS UNI - Byte in 10	~~%IB55~~	BYTE

Fig. 20-40: Mapping process data of RS UNI on sercos bus

20.17 IH_MBWriteFileRecord

Function code FC 21

Brief description The function block **IH_MBWriteFileRecord** is used to write a file record in RTU mode. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-47: Reference table

Interface description

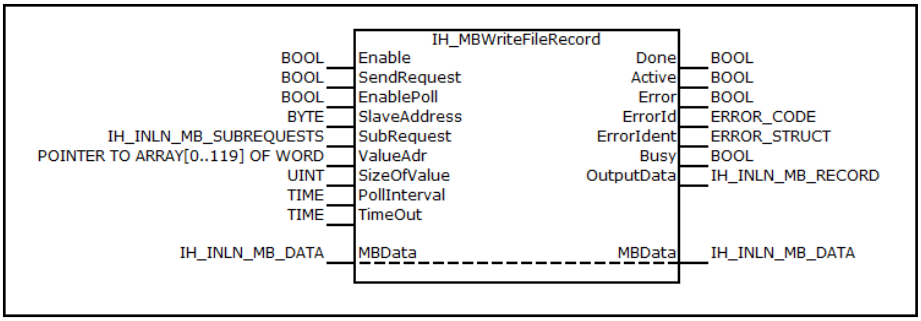


Fig. 20-41: IH_MBWriteFileRecord function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	SubRequest	IH_INLN_MB_SUBREQUESTS	Sub request. IH_INLN_MB_SUBREQUESTS, page 866
	ValueAdr	POINTER TO ARRAY[0..119] OF WORD	Address of the Record Data Array. Use ADR () operator
	SizeOfValue	UINT	Size of the ValueAdr array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
	OutputData	IH_INLN_MB_RECORD	It contains Request data length and Sub requested data. Please refer to: Structures and enumerations IH_INLN_MB_RECORD, page 866
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w the Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-48: Interface variables of the IH_MBWriteFileRecord function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
SizeOfValue	UINT	1	----------	----------	Continuous with Enable

Name	Type	Min. value	Max. value	Default value	Effective
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-49: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBWriteFileRecord

Functional description

The function block **IH_MBWriteFileRecord** (FC 21) is used to write a file record. All Request Data Length in the structure (IH_INLN_MB_SUBREQUESTS) are provided in terms of number of bytes and all Record Lengths are provided in terms of the number of words. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program; its structure communicates and exchanges the data.

The function will write a single group of references. The groups can be separating (noncontiguous),

but the references within each group must be sequential.

Each group is defined in a separate "sub-request" field that contains 7 bytes:

- The reference type: 1 byte (must be specified as 6). NOTE: Considered default in this FB
- The file number: (Word) should be (0x0001 to 0xFFFF)
- The Start record number within the file: (word) should be (0x0000 to 0x270F)
- The length of the record to be read: (Word) Should be (1 to 120)

The quantity of registers to be read, combined with all other fields in the expected response,

must not exceed the allowable length of the MODBUS PDU : 253 bytes.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the IH_MBWriteFileRecord function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBWriteFileRecord function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBWriteFileRecord function block is used to write the file record to the slave device.

Write file record with SendRequest condition

The **IH_MBWriteFileRecord** used to write Record of Record length 10 from Record num 4 in the file num 1 of slave Address 2 using "bSendRequest" as BOOL to trigger the write file.

Variable declaration

```
VAR
    fbIH_MBWriteFileRecord: IH_MBWriteFileRecord;
    bEnable: BOOL;
```

```
    bSendRequest: BOOL;
    stSubRequest: IH_INLN_MB_SUBREQUESTS;
    arWriteRecordData: ARRAY [0..119] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
    stOutputData: IH_INLN_MB_RECORD;
END_VAR
```

Implementation

```
stSubRequest.FileNumber := WORD#1;
stSubRequest.RecordNumber := WORD#4;
stSubRequest.RecordLength := WORD#10;


fbIH_MBWriteFileRecord(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    SubRequest:= stSubRequest,
    ValueAdr:= ADR(arWriteRecordData),
    SizeOfValue:= SIZEOF(arWriteRecordData),
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy,
    OutputData=> stOutputData);
```

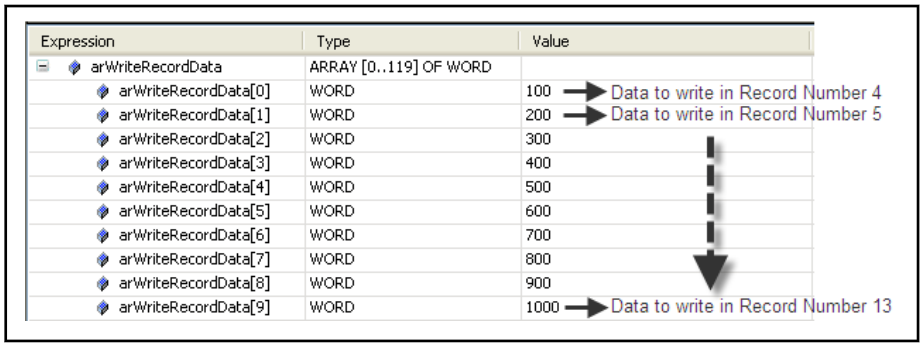


Fig. 20-42: arWriteRecordData for program 1

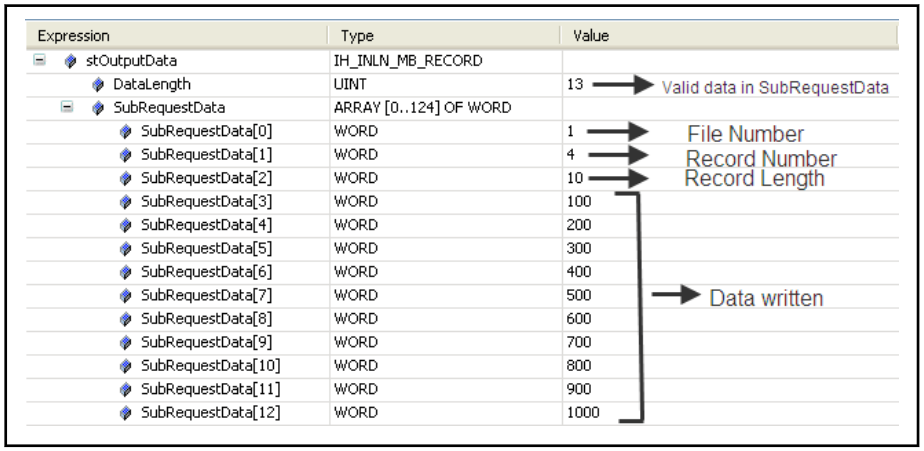


Fig. 20-43: stOutputData for program 1

Write file record with Poll Interval condition

The **IH_MBWriteFileRecord** used to write Record of Record length 5 from Record num 1 in the file num 2 of slave Address 5 using Poll interval (3 seconds) to trigger the write file.

Variable declaration

```
VAR
    fbIH_MBWriteFileRecord2: IH_MBWriteFileRecord;
    bEnable2: BOOL;
    stSubRequest2: IH_INLN_MB_SUBREQUESTS;
 stMBData: IH_INLN_MB_DATA;
    arWriteRecordData2: ARRAY [0..4] OF WORD;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
    stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
    stOutputData2: IH_INLN_MB_RECORD;
END_VAR
```

Implementation

```
stSubRequest2.FileNumber := WORD#2;
stSubRequest2.RecordNumber := WORD#1;
stSubRequest2.RecordLength := WORD#5;


fbIH_MBWriteFileRecord2(
    Enable:= bEnable2,
    SendRequest:= FALSE,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#5,
    SubRequest:= stSubRequest2,
    ValueAdr:= ADR(arWriteRecordData2),
    SizeOfValue:= SIZEOF(arWriteRecordData2),
    PollInterval:= T#3S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
    ErrorID=> enErrorID2,
    ErrorIdent=> stErrorIdent2,
    Busy=> bBusy2,
    OutputData=> stOutputData2);
```

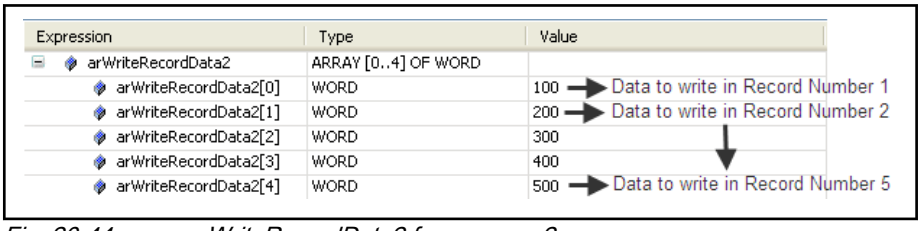


Fig. 20-44: arWriteRecordData2 for program 2

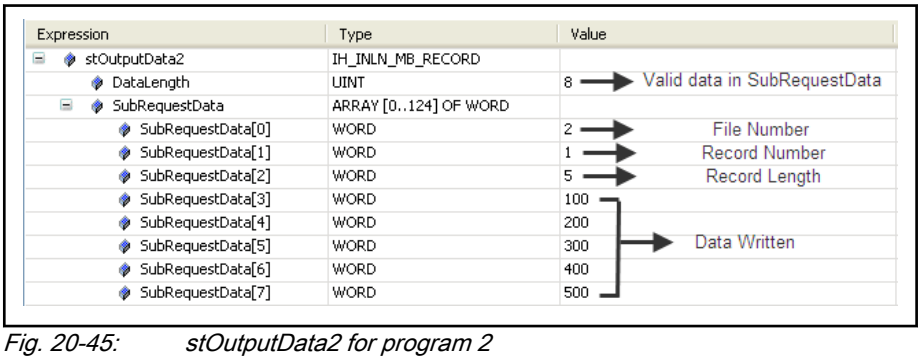


Fig. 20-45: stOutputData2 for program 2

20.18 IH_MBWriteMultipleCoils

Function code FC 15

Brief description The function block **IH_MBWriteMultipleCoils** is used to force multiple coils, i.e. output bits to either ON or OFF of a slave module. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-50: Reference table

Interface description

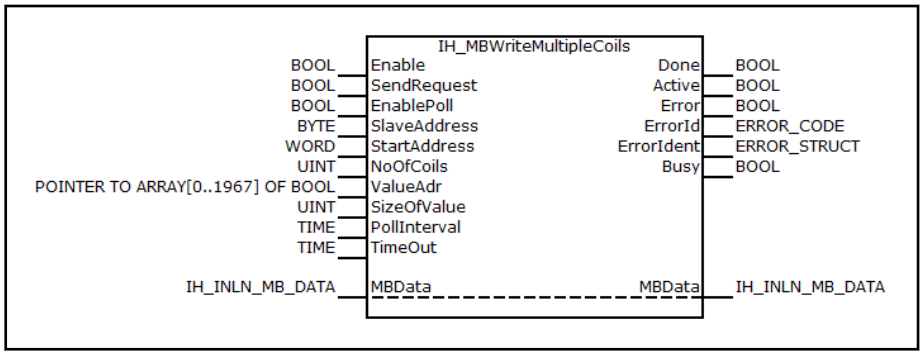


Fig. 20-46: IH_MBWriteMultipleCoils function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of output bits.
	NoOfCoils	UINT	Quantity of coils (discrete outputs)
	ValueAdr	POINTER TO ARRAY[0..1967] OF BOOL	Address of the array to store the write value of the output bits. Use ADR () operator
	SizeOfValue	UINT	Size of the WriteDataVal array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond

I/O type	Name	Data type	Description
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-51: Interface variables of the IH_MBWriteMultipleCoils function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	FFFF	0	Rising edge at Enable
NoOfCoils	UINT	1	1968	1	Rising edge at Enable
SizeOfValue	UINT	1	1968	----------	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-52: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBWriteMultipleCoils

Functional description

The function block **IH_MBWriteMultipleCoils** (FC 15) is used to force multiple coils, i.e. output bits to either ON or OFF of a slave module. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program, its structure communicates and exchanges the data.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the IH_MBWriteMultipleCoils function block to communicate with the Modbus

slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBWriteMultipleCoils function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBWriteMultipleCoils function block is used to write coils to the slave device.

Write coils with SendRequest condition

IH_MBWriteMultipleCoils is used to write 7 coils of the Start address 0x1000 to slave Address 2 using the "bSendRequest" as BOOL to trigger the write coils.

Variable declaration

```
VAR
    fbIH_MBWriteMultipleCoils: IH_MBWriteMultipleCoils;
    bEnable: BOOL;
    bSendRequest: BOOL;
    arWriteMultipleCoils: ARRAY [0..6] OF BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBWriteMultipleCoils(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#1000,
    NoOfCoils:= UINT#7,
    ValueAdr:= ADR(arWriteMultipleCoils),
    SizeOfValue:= SIZEOF(arWriteMultipleCoils),
    PollInterval:= T#2S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

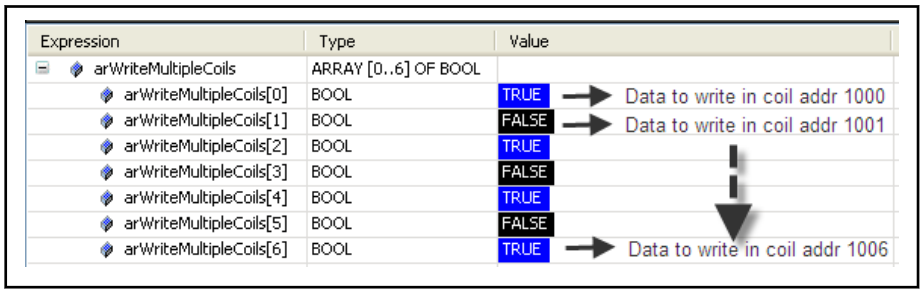


Fig. 20-47: Data which needs to be transferred to slave device in program 1

Write coils with Poll interval condition

IH_MBWriteMultipleCoils is used to write 4 coils of the Start address 0x0500 to slave Address 5 using the "Poll Interval" (3 seconds) to trigger the write coils.

Variable declaration

```
VAR
    fbIH_MBWriteMultipleCoils2: IH_MBWriteMultipleCoils;
    bEnable2: BOOL;
```

```
    arWriteMultipleCoils2: ARRAY [0..3] OF BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    stErrorIdent2: ERROR_STRUCT;
    enErrorID2: ERROR_CODE;
    bBusy2: BOOL;
END_VAR
```

Implementation

```
fbIH_MBWriteMultipleCoils2(
    Enable:= bEnable2,
    SendRequest:= FALSE,
    EnablePoll:= TRUE,
    SlaveAddress:= BYTE#5,
    StartAddress:= WORD#500,
    NoOfCoils:= UINT#4,
    ValueAdr:= ADR(arWriteMultipleCoils2),
    SizeOfValue:= SIZEOF(arWriteMultipleCoils2),
    PollInterval:= T#3S,
    TimeOut:= T#2S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
    ErrorID=> enErrorID2,
    ErrorIdent=> stErrorIdent2,
    Busy=> bBusy2);
```

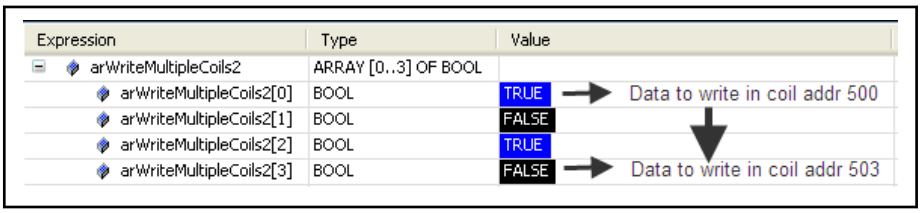


Fig. 20-48: Data which needs to be transferred to the slave device in program 2

20.19 IH_MBWriteMultipleRegisters

Function code FC 16

Brief description The function block **IH_MBWriteMultipleRegisters** is used to write holding registers of the Modbus slave module. The function block should be called with the Modbus master function IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-53: Reference table

Interface description

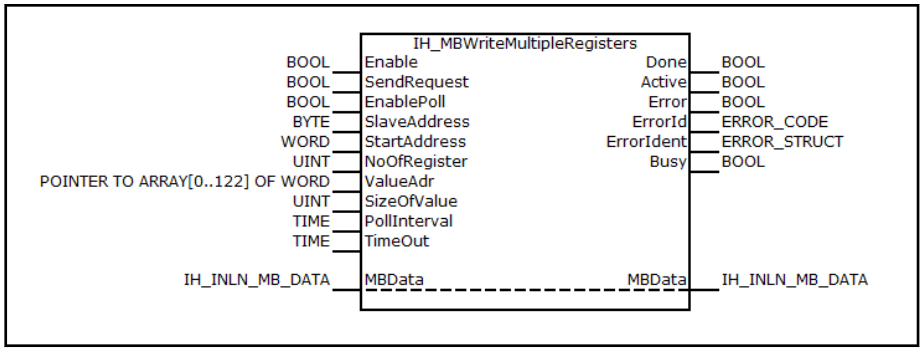


Fig. 20-49: IH_MBWriteMultipleRegisters function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for **SendRequest** to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of holding register
	NoOfRegisters	UINT	Quantity of registers
	ValueAdr	POINTER TO ARRAY[0..122] OF WORD	Address of the Array to contains write value of the registers. Use ADR () operator
	SizeOfValue	UINT	Size of the Value array. Use SIZEOF () operator
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-54: Interface variables of the IH_MBWriteMultipleRegisters function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	65535	0	Rising edge at Enable
NoOfRegisters	UINT	1	123	1	Rising edge at Enable
SizeOfValue	UINT	1	----------	----------	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-55: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBWriteMultipleRegisters

Functional description

The function block **IH_MBWriteMultipleRegisters** (FC 16) is used to write holding registers of the slave module specified at the SlaveAddress input. The function block should be called with the Modbus master function block. "MBData" should link to the master function block to communicate and exchanges the data with the master.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The function block IH_MBWriteMultipleRegisters should be instantiated along with the IH_MBRTUMaster to communicate with the Modbus slave. Use a common variable at the MBData of the IH_ MBWriteMultipleRegisters and IH_MBRTUMaster function blocks to exchange the data with the Modbus slave.

Example 1

In this example, the function block IH_ MBWriteMultipleRegisters is used to write the holding register of slave address 1 with StartAddress 2000 and NoOfRegisters 5.

Enable both IH_MBRTUMaster and IH_ MBWriteMultipleRegisters function blocks, with the rising edge at bSendRequest input, the function block writes the data to slave address 1 and after successful completion of the write operation, the done bit is set to true for one PLC cycle.

Variable declaration

```
VAR
    fbIH_MBWriteMultipleRegisters: IH_MBWriteMultipleRegisters;
    bEnable: BOOL;
    bSendRequest: BOOL;
    arWriteRegisters: ARRAY [0..4] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
```

```
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBWriteMultipleRegisters(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#1,
    StartAddress:= WORD#2000,
    NoOfRegisters:= UINT#5,
    ValueAdr:= ADR(arWriteRegisters),
    SizeOfValue:= SIZEOF(arWriteRegisters),
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

Request

The data to be written to the slave module should be entered in the arWriteRegisters array.

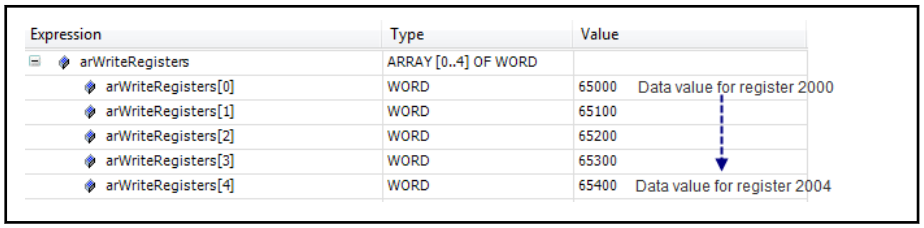


Fig. 20-50: arWriteRegisters array

Example 2

In this example, the function block IH_ MBWriteMultipleRegisters is used to write the holding register of slave address 2 with StartAddress 3000 and NoOfRegisters 10 with a poll interval of 1 second. Enable both IH_MBRTUMaster and IH_ MBWriteMultipleRegisters function blocks and set EnablePoll to true with a PollInterval of 1 second. Subsequently, the function block writes data to slave address 2 with a regular interval of 1 second and after each successful completion of the write operation, the done bit is set to true for one PLC cycle.

Variable declaration

```
VAR
    fbIH_MBWriteMultipleRegisters2: IH_MBWriteMultipleRegisters;
    bEnable2: BOOL;
    bSendRequest2: BOOL;
    bEnablePoll: BOOL;
    arWriteRegisters2: ARRAY [0..9] OF WORD;
    stMBData: IH_INLN_MB_DATA;
    bDone2: BOOL;
    bActive2: BOOL;
    bError2: BOOL;
    enErrorID2: ERROR_CODE;
    stErrorIdent2: ERROR_STRUCT;
    bBusy2: BOOL;
END_VAR
```

Implementation

```
fbIH_MBWriteMultipleRegisters2(
    Enable:= bEnable2,
    SendRequest:= bSendRequest2,
    EnablePoll:= bEnablePoll,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#3000,
    NoOfRegisters:= UINT#10,
    ValueAdr:= ADR(arWriteRegisters2),
    SizeOfValue:= SIZEOF(arWriteRegisters2),
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone2,
    Active=> bActive2,
    Error=> bError2,
    ErrorID=> enErrorID2,
    ErrorIdent=> stErrorIdent2,
    Busy=> bBusy2);
```

Response:

The data to be written to the slave module should be entered in the arWriteRegisters2 array.

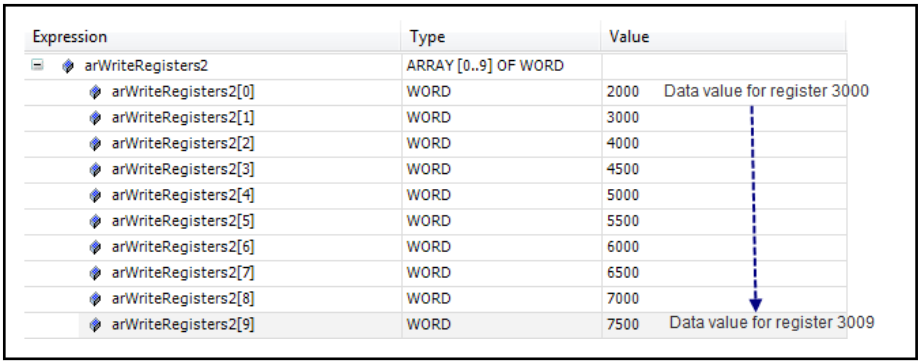


Fig. 20-51: arWriteRegisters2 array

20.20 IH_MBWriteSingleCoil

Function code FC 5

Brief description The function block **IH_MBWriteSingleCoil** is used to write the ON or OFF value to a single coil, i.e. a single output bit of the slave module. The function block should be called with the Modbus master function block IH_MBRTU-Master, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-56: Reference table

Interface description

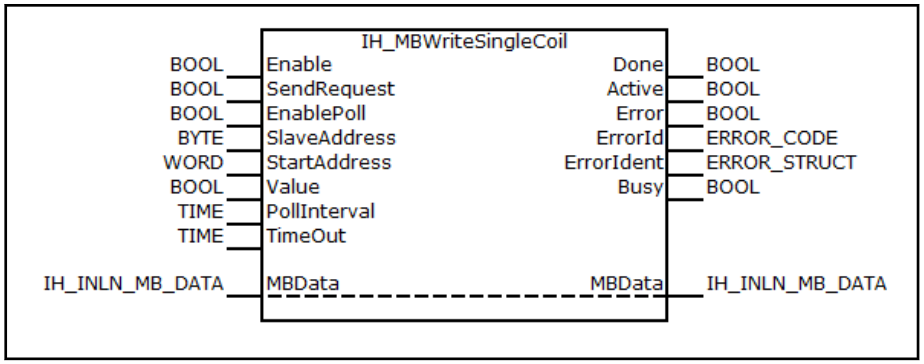


Fig. 20-52: IH_MBWriteSingleCoil function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of input bits
	Value	BOOL	Output bit value: True or False
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-57: Interface variables of the IH_MBWriteSingleCoil function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable

Name	Type	Min. value	Max. value	Default value	Effective
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	FFFF	0	Rising edge at Enable
Value	BOOL	FALSE	TRUE	FALSE	Rising edge at Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-58: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBWriteSingleCoil

Functional description

The function block **IH_MBWriteSingleCoil** (FC 5) is used to write the ON or OFF Value to a single coil, i.e. a single output bit of the slave module. The function block should be called with the Modbus master function block. "MBData" should link to the master function block in the program; its structure communicates and exchanges the data.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The IH_MBRTUMaster function block should be instantiated along with the IH_MBWriteSingleCoil function block to communicate with the Modbus slave. The in/out variable stMBData exchanges the data between the IH_MBRTUMaster and IH_MBWriteSingleCoil function blocks, so the same variable should be assigned to MBData in both function blocks.

In the example, the IH_MBWriteSingleCoil function block is used to write a single coil in the slave device.

Writing single coil with SendRequest condition

IH_MBWriteSingleCoil is used to write a single coil of the start address 0x1000 to slave Address 2 using "bSendRequest" as BOOL to trigger write single coil.

Variable declaration

```
VAR
    fbIH_MBWriteSingleCoil: IH_MBWriteSingleCoil;
    bEnable: BOOL;
    bSendRequest: BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBWriteSingleCoil(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= FALSE,
    SlaveAddress:= BYTE#2,
    StartAddress:= WORD#1000,
    Value:= TRUE,
```

```
PollInterval:= T#2S,
TimeOut:= T#2S,
MBData:= stMBData,
Done=> bDone,
Active=> bActive,
Error=> bError,
ErrorID=> enErrorID,
ErrorIdent=> stErrorIdent,
Busy=> bBusy);
```

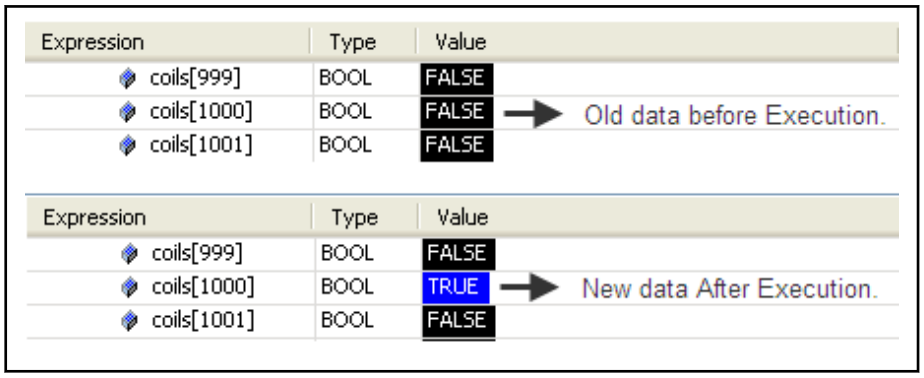


Fig. 20-53: Shows the modification after execution

20.21 IH_MBWriteSingleRegister

Function code FC 6

Brief description The function block **IH_MBWriteSingleRegister** is used to write the single holding register of the Modbus to the slave module. The function block should be called with the Modbus master function block IH_MBRTUMaster, page 835.

Target system	Library / Folder
MLC	RIH_Modbus / FB
MTX	RIH_Modbus / FB
MLD (IL 2G)	RIH_Modbus / FB

Tab. 20-59: Reference table

Interface description

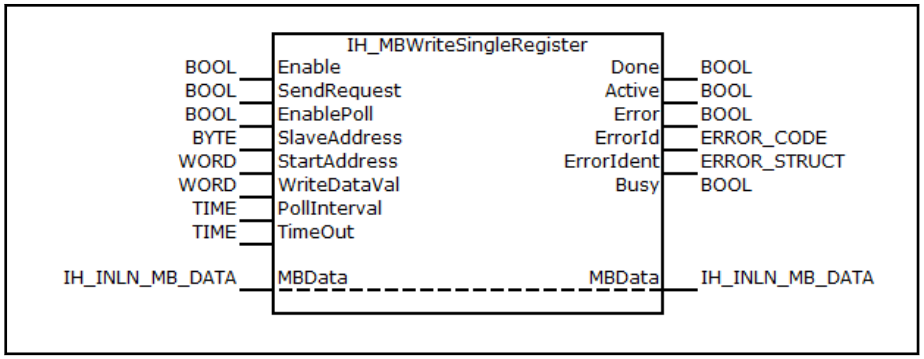


Fig. 20-54: IH_MBWriteSingleRegister function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block is running when true
	SendRequest	BOOL	A rising edge sends a request
	EnablePoll	BOOL	Polling enabled. No need for SendRequest to be triggered every interval. The block sends requests automatically
	SlaveAddress	BYTE	Slave address
	StartAddress	WORD	Start address of holding register
	WriteDataVal	WORD	Write value of the register.
	PollInterval	TIME	Polling interval. Request is sent continuously in intervals
	TimeOut	TIME	Timeout interval for slaves to respond
VAR_OUTPUT	Done	BOOL	Calculation finished and will restart in next cycle
	Active	BOOL	Function block is active
	Error	BOOL	Indicates an error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information Error Handling of Library RIH_Modbus, page 867
	Busy	BOOL	True when the block is communicating with master
VAR_IN_OUT	MBData	IH_INLN_MB_DATA	Structure used to handle data b/w Master and other Modbus function blocks of function codes IH_INLN_MB_DATA, page 864

Tab. 20-60: Interface variables of the IH_MBWriteSingleRegister function block

Min./max. and default values

The following table lists the min./max. and default values as well as the transfer behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
EnablePoll	BOOL	FALSE	TRUE	FALSE	Continuous with Enable
SlaveAddress	BYTE	0	247	0	Rising edge at Enable
StartAddress	WORD	0	65535	0	Rising edge at Enable
WriteDataVal	WORD	0	65535	0	Continuous with Enable
PollInterval	TIME	T#1s	----------	T#1s	Continuous with Enable
TimeOut	TIME	T#1s	----------	T#1s	Rising edge at Enable

Tab. 20-61: Min., max. and default values as well as the transfer behavior of the inputs of the function block IH_MBWriteSingleRegister

Functional description

The function block **IH_MBWriteSingleRegister** (FC 6) is used to write the single holding register of the Modbus slave module specified at the SlaveAddress input. The function block should be called with the Modbus master function block. "MBData" should link to the master function block to communicate and exchanges the data with the master.

Activate the function block after the first PLC cycle by setting the "Enable" input to TRUE. During the operation, the "Active" output is set to TRUE. Upon completion, the "Done" bit is set to TRUE. If an error occurs during the operation, the error bit is set to TRUE. ErrorID and ErrorIdent display the error information. The error can then be acknowledged with a rising edge at the "Enable" input.

Example of a function block

The function block IH_MBWriteSingleRegister should be instantiated along with the IH_MBRTUMaster to communicate with the Modbus slave. Use a common variable at the MBData of IH_ MBWriteSingleRegister and IH_MBRTUMaster function blocks to exchange the data with the Modbus slave.

In this example, the function block IH_ MBWriteSingleRegister is used to write the holding register of slave address 1 with StartAddress 1000 and WriteDataVal 200.

Enable both IH_MBRTUMaster and IH_ MBWriteSingleRegister function blocks, with the rising edge at the bSendRequest input, the function block writes the data to slave address 1.

By setting the bEnablePoll variable to TRUE and Pollinterval to more than 1 second, the data is written to the register with a regular interval.

After each successful completion of the write operation, the done bit is set to true for one PLC cycle.

Variable declaration

```
VAR
    fbIH_MBWriteSingleRegister: IH_MBWriteSingleRegister;
    bEnable: BOOL;
    bSendRequest: BOOL;
    bEnablePoll: BOOL;
    stMBData: IH_INLN_MB_DATA;
    bDone: BOOL;
    bActive: BOOL;
    bError: BOOL;
    enErrorID: ERROR_CODE;
    stErrorIdent: ERROR_STRUCT;
    bBusy: BOOL;
END_VAR
```

Implementation

```
fbIH_MBWriteSingleRegister(
    Enable:= bEnable,
    SendRequest:= bSendRequest,
    EnablePoll:= bEnablePoll,
    SlaveAddress:= BYTE#1,
    StartAddress:= WORD#1000,
    WriteDataVal:= WORD#200,
    PollInterval:= T#1S,
    TimeOut:= T#1S,
    MBData:= stMBData,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    Busy=> bBusy);
```

20.22 Data types

20.22.1 IH_INLN_MB_DATA

The structure used internally to exchange the command and the data between the master function blocks and other function blocks of function codes.

Library	Folder
RIH_Modbus	Type

Tab. 20-62: Reference table

Name	Type	Description	Min. value	Max. value	Default value	Application
Ready	BOOL	Master FB is ready to communicate	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	Indicates master wants to send request.	FALSE	TRUE	FALSE	Continuous
SlaveAddress	BYTE	Address of the concerned slave	0	247	0	Continuous
FunctionCode	INT	Function code queried by the master	-------	-------	-------	Continuous
StartAddress	WORD	Start address for code execution	--------	-------	--------	Continuous
ArrayData	ARRAY[0..124] OF WORD.	Data bytes of the telegram	--------	-------	--------	Continuous
TimeOut	TIME	Time out interval	T#1s	-------	T#1s	Continuous
RespErr	BYTE	Error information	--------	-------	--------	Continuous
Busy	BOOL	Master block is busy	FALSE	TRUE	FALSE	Continuous
NewDataRecv	BOOL	New data is received	FALSE	TRUE	FALSE	Continuous
DataCount	UINT	Data length	--------	-------	--------	Continuous
Done	BOOL	Process (request and response) is complete	FALSE	TRUE	FALSE	Continuous
Reset	BOOL	Internal reset variable used to reset master	FALSE	TRUE	FALSE	Continuous
Token	ARRAY[0..32] OF IH_INLN_MB_TOKEN	Token	--------	-------	--------	Continuous

Tab. 20-63: IH_INLN_MB_DATA

20.22.2 IH_INLN_MB_DEV_IDCODE

The ENUM **IH_INLN_MB_DEV_IDCODE** specifies the object category access levels. Function block IH_MBReadDeviceIdentification on page 801 supports the following Read Device Id code access types:

Name	Value	Description
IH_INLN_MB_READ01	1	request to get the basic device identification (stream access)
IH_INLN_MB_READ02	2	request to get the regular device identification (stream access)

Name	Value	Description
IH_INLN_MB_READ03	3	request to get the extended device identification (stream access)
IH_INLN_MB_READ04	4	request to get one specific identification object (individual access)

For an individual object id access "DeviceIdCode" as **IH_INLN_MB_READ04**, desired object id should be specified in input "ObjectId". For the rest of device id codes, "ObjectId" should be 0.

20.22.3 IH_INLN_MB_FIFODATA

The structure used to represent the final response output of **IH_MBReadFIFOQue** function block (IH_MBReadFIFOQue, page 816).

Library	Folder
RIH_Modbus	Type

Tab. 20-64: Reference table

Name	Type	Description
ByteCount	UINT	Byte count (total no of received bytes)
FIFOCount	UINT	Quantity of FIFO registers
FIFORegValues	ARRAY[1..31] OF WORD	FIFO registers values

20.22.4 IH_INLN_MB_CONFIG

The structure is the extension of IL_INLN_RS_CONFIG used to configure the communication settings of the Modbus master/slave function block (RIL_InlineIOModules.library, FB IL_RsCom, STRUCT IL_INLN_RS_CONFIG).

Library	Folder
RIH_Modbus	Type

Tab. 20-65: Reference table

Name	Type	Description	Min. value	Max. value	Default value	Application
Protocol	IL_INLN_RS_PROTOCOL	Not applicable to Modbus.				
Baudrate	IL_INLN_RS_BAUDRATE	Baud rate. For documentation please refer to: IL_INLN_RS_BAUDRATE	BAUDRATE_110	BAUDRATE_250000	BAUDRATE_9600	Rising edge at Enable
DataWidth	IL_INLN_RS_DATAWIDTH	Data width	DATABIT7_EVEN_STOPBIT1	DATABIT6_NONE_STOPBIT1	DATABIT7_EVEN_STOPBIT1	Rising edge at Enable
ErrorPattern	BYTE	Not applicable to Modbus				
FirstDelimiter	BYTE	Not applicable to Modbus				
SecondDelimiter	BYTE	Not applicable to Modbus				

Name	Type	Description	Min. value	Max. value	Default value	Application
InterfaceType	IL_INLN_RS_INTERFACE	Type of interface (RS232, RS485 or RS422). For documentation please refer to: IL_INLN_RS_INTERFACE	RS232	RS422	RS232	Rising edge at Enable
ModuleType	IL_INLN_RS_MODULETYPE	Used to select connected module. For documentation please refer to: IL_INLN_RS_MODULETYPE	RS232PRO	RSUNI_31WORDS	RS232PRO	Rising edge at Enable

Tab. 20-66: IH_INLN_MB_CONFIG

20.22.5 IH_INLN_MB_TOKEN

The structure used to get the Token availability of master function block.

Library	Folder
RIH_Modbus	Type

Tab. 20-67: Reference table

Name	Data type	Description	Min. value	Max. value	Default value	Application
Occupied	BOOL	If token number is already used	FALSE	TRUE	FALSE	Continuous
SendRequest	BOOL	Request sent to master for token	FALSE	TRUE	FALSE	Continuous
RequestGranted	BOOL	Request granted by master for token access	FALSE	TRUE	FALSE	Continuous

Tab. 20-68: IH_INLN_MB_TOKEN

20.22.6 IH_INLN_MB_SUBREQUESTS

The structure used to get the details to access the Modbus File record data.

Library	Folder
RIH_Modbus	Type

Tab. 20-69: Reference table

Name	Data type	Description
FileNumber	WORD	File number
RecordNumber	WORD	Record number
RecordLength	WORD	Record length

Tab. 20-70: IH_INLN_MB_SUBREQUESTS

20.22.7 IH_INLN_MB_RECORD

The structure used to represent the final response output of **IH_MBReadFileRecord** function block

Library	Folder
RIH_Modbus	Type

Tab. 20-71: Reference table

Name	Data type	Description
DataLength	UINT	Total no of data received (output)
SubRequestData	ARRAY[0..124] OF WORD	Requested record data

Tab. 20-72: IH_INLN_MB_RECORD

20.22.8 IH_INLN_MB_EVENTLOG

The structure used to represent the final response output of **IH_MBGetCommEventLog** function block.

Library	Folder
RIH_Modbus	Type

Tab. 20-73: Reference table

Name	Data type	Description
Status	WORD	Status
EventCount	WORD	Event count
MessageCount	WORD	Message count
Events	ARRAY[0..63] OF BYTE	Events

Tab. 20-74: IH_INLN_MB_EVENTLOG

20.23 Error Handling of Library RIH_Modbus

The function blocks use the error table **INLINEIO_TABLE**, 16#0190. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#3002	16#0001	The Baud rate is not valid
INPUT_INVALID_ERROR	16#3002	16#0002	Data width is not valid
INPUT_INVALID_ERROR	16#3002	16#0003	Protocol is not valid
INPUT_INVALID_ERROR	16#3002	16#0004	Interface input is not valid
INPUT_INVALID_ERROR	16#3002	16#0005	Size of send data is not valid
INPUT_INVALID_ERROR	16#3002	16#0006	Size of recv data is not valid
INPUT_INVALID_ERROR	16#3002	16#0007	SizeOfInput or SizeOfOutput is not valid
INPUT_INVALID_ERROR	16#3010	16#0000	Incorrect module type is connected
INPUT_INVALID_ERROR	16#3022	16#0000	‘SlaveAddress’ input value is invalid (0 < SlaveAddress > 247)
INPUT_INVALID_ERROR	16#3022	16#0001	‘NoOfInputBits/NoOfCoils’ input value is invalid For Reading: (1 < NoOfInputBits/NoOfCoils > 2000) For Writing: (1 < NoOfInputBits/NoOfCoils > 1968)

INPUT_INVALID_ERROR	16#3022	16#0002	‘SizeOfReadDataVal’ input is invalid (SizeOfReadDataVal < NoOfReadRegisters)
INPUT_INVALID_ERROR	16#3022	16#0003	‘NoOfRegisters’ input is invalid For Reading: (1 < NoOfRegisters > 125) For Writing: (1 < NoOfRegisters > 123)
INPUT_INVALID_ERROR	16#3022	16#0004	’NoOfSubRequests’ input is invalid (1 < NoOfSubRequests > 30)
INPUT_INVALID_ERROR	16#3022	16#0005	‘RecordNumber’ input is invalid (0 < IH_INLN_MB_SUBREQUESTS.RecordNumber > 9999)
INPUT_INVALID_ERROR	16#3022	16#0006	‘FileNumber’ input is invalid (FileNumber < 1)
INPUT_INVALID_ERROR	16#3022	16#0007	‘RecordLength’ input is invalid (1 < IH_INLN_MB_SUBREQUESTS.RecordLength > 120
INPUT_INVALID_ERROR	16#3022	16#0008	Total record length of all SubRequests should be less than (123 – NoOfSubRequests)
INPUT_INVALID_ERROR	16#3022	16#0009	‘SizeOfWriteDataVal’ input is invalid (SizeOfWriteDataVal < NoOfWriteRegisters)
INPUT_INVALID_ERROR	16#3022	16#000A	‘PollInterval/TimeOut’ input is invalid (If it is less than 1 sec)
INPUT_INVALID_ERROR	16#3022	16#000C	‘SizeOfValue’ input is invalid (SizeOfValue < ‘NoOfCoils/NoOfRegisters/NoOfInputBits/RecordLength’)
INPUT_INVALID_ERROR	16#3022	16#000D	‘NoOfReadRegisters’ input is invalid (1 < NoOfReadRegisters > 125)
INPUT_INVALID_ERROR	16#3022	16#000E	‘NoOfWriteRegisters’ input is invalid (1 < NoOfWriteRegisters > 121)
INPUT_INVALID_ERROR	16#3022	16#000F	‘SubFunc’ input is invalid. Please refer to the supported Sub-Function Codes of read diagnostics
INPUT_INVALID_ERROR	16#3022	16#0010	SizeOfCoils input value is invalid (1 < SizeOfCoils > 8191)
INPUT_INVALID_ERROR	16#3022	16#0011	SizeOfInputBits input value is invalid (1 < SizeOfInputBits > 8191)
INPUT_INVALID_ERROR	16#3022	16#0012	SizeOfHoldingRegisters input value is invalid (1 < SizeOfHoldingRegisters > 65535)
INPUT_INVALID_ERROR	16#3022	16#0013	SizeOfInputRegisters input value is invalid (1 < SizeOfInputRegisters > 65535)
DEVICE_ERROR	16#3020	16#0000	Timeout error. Device not responding
DEVICE_ERROR	16#3023	16#0000	Timeout error. Device not responding
DEVICE_ERROR	16#3023	16#0001	Modbus Exception Code 16#01, illegal function
DEVICE_ERROR	16#3023	16#0002	Modbus Exception Code 16#02, illegal data address
DEVICE_ERROR	16#3023	16#0003	Modbus Exception Code 16#03, illegal data value
DEVICE_ERROR	16#3023	16#0004	Modbus Exception Code 16#04, Slave device failure
DEVICE_ERROR	16#3023	16#0005	Modbus Exception Code 16#05, Acknowledge
DEVICE_ERROR	16#3023	16#0006	Modbus Exception Code 16#06, Slave device busy
DEVICE_ERROR	16#3023	16#0008	Modbus Exception Code 16#08, Memory parity error
DEVICE_ERROR	16#3023	16#000A	Modbus Exception Code 16#0A, Gateway path unavailable

DEVICE_ERROR	16#3023	16#000B	Modbus Exception Code 16#0B, Gateway target device failed to respond
DEVICE_ERROR	16#3023	16#000C	Different CRC values for request and response
DEVICE_ERROR	16#3023	16#000D	Invalid response, this error may occur when master and slave are communicating at different speed, or slave sends a non-Modbus response
COMMUNICATION_ERROR	16#3011	16#0000	Timeout during sending
COMMUNICATION_ERROR	16#3011	16#0001	Error while sending data
COMMUNICATION_ERROR	16#3011	16#0002	Error while sending data to buffer
COMMUNICATION_ERROR	16#3012	16#0000	Timeout during receiving
RESOURCE_ERROR	16#3013	16#0000	Null pointer, variable not assigned
OTHER_ERROR	16#3024	16#0000	Master function block is not ready
INPUT_RANGE_ERROR	16#3025	16#0000	Number of function code FB’s exceeded for master. (Maximum FB’s = 33). Token not available for processing

Tab. 20-75: RIH_Modbus error codes

21 RIL_IoLink.library

21.1 Introduction and overview

The library RIL_IoLink includes field bus-independent function blocks to control IO-Link devices. For field bus-dependent function blocks to control IO-Link devices, refer to the library on the respective field bus, e.g. RIL_ProfinetIO.library.

Target systems

The library can be used in the following systems:

Target system	Library
MLC/MTX	RIL_IoLink
MLD with IndraLogic 2G ≥ 20VRS	RIL_IoLink

Areas of application and function blocks of the RIL_IoLink

Area of application (folder in library)	
Function blocks	**Description**
IL_SIII_IoLinkProxy, page 873	The function block implements the proxy function block to communicate to the IO-Link master via Sercos
IL_IoLinkCall, page 875	The function block IL_IoLinkCall implements the client function block for the IO-Link communication to the IO-Link master
IL_IoLinkErrorText, page 880	Function converting the error codes of the IO-Link function blocks into readable text
IL_IoLinkIdent, page 880	This function block queries all mandatory IO-Link parameters required to identify a device
RexrothHydraulicValve (controlling hydraulic IO-Link devices)	
IL_IoLinkDirectValveControlType01, page 882	The function block is used to control a Bosch Rexroth IO-Link valve
With data types:	
IL_IOL_OUT_PROC_SPOOL_TYPE01, page 885	Process output data of the function block IL_IoLinkDirectValveControlType01
IL_IOL_IN_PROC_SPOOL_TYPE01, page 886	Process input data of the function block IL_IoLinkDirectValveControlType01

Tab. 21-1: RIL_IoLink library

21.2 General information on the IO-Link

IO-Link is a standard to connect sensors/actuators on field level. There is a point-to-point connection between the IO-Link master and the IO-Link device. IO-Link masters generally show a gateway functionality to existing field buses. Apart from cyclic process data, acyclic data can be read for configuration and identification purposes. These are provided in the IO-Link master and in the connected IO-Link devices. The following function blocks are used to read and write these parameters.

Sercos-specific IO-Link design

To read/write IO-Link parameters using the Sercos field bus, Sercos has to communicate with the IO-Link master. The IO-Link master is a Sercos slave in this case. Sercos tunnels the IO-Link communication. The specific Sercos sub-bus protocol is used. The parameters S-0-1508.x.65, S-0-1508.x.66 and S-0-1508.x.67 are used to communicate with the IO-Link master. Generally, multiple IO-Link devices are connected to one IO-Link master. Thus, multiple communication channels are required. Only one request can be processed at a time via the Sercos field bus. Thus, ensure that not multiple requests are sent to the IO-Link master at the same time. A proxy concept is implemented. It is described in the following.

The proxy function block communicates (Sercos-specifically) with the IO-Link master. This proxy function block may only be instantiated once per Sercos slave. An incorrect multiple instantiation cannot be avoided. The IO-Link client function block uses only IO-Link-specific data and communicates with the proxy. Multiple instances of the client can be present. These instances have to register at the proxy and can then exchange data independently with the IO-Link device. The proxy ensures that all requests of the client sent to the IO-Link master are serialized and that the responses are returned to the respective client. The user does not have to implement the mechanisms to allow the communication between the client and the proxy. These are internal mechanisms.

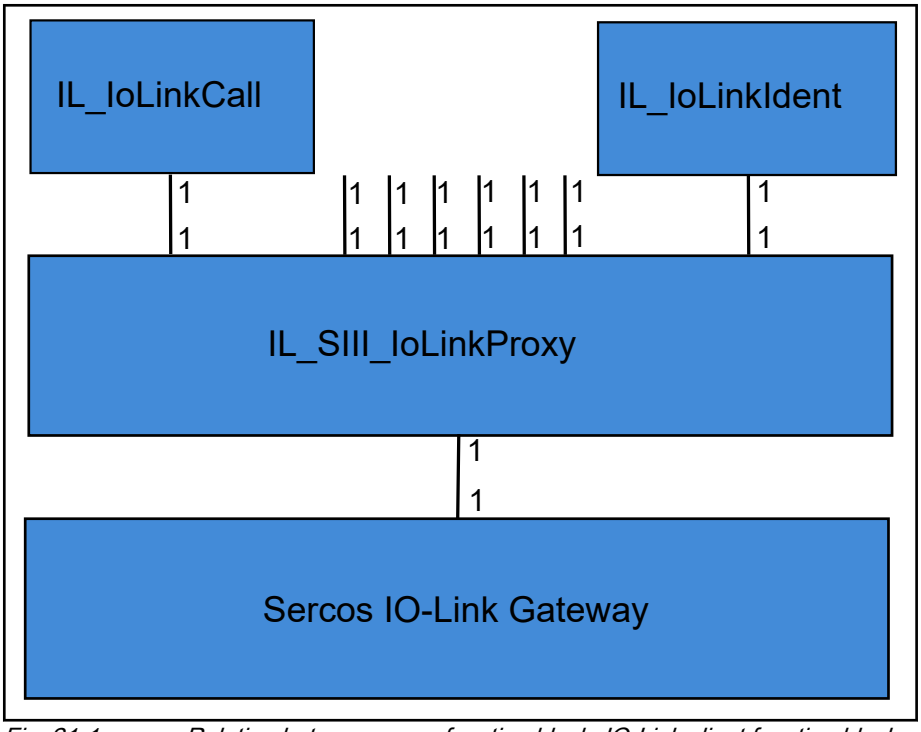


Fig. 21-1: Relation between proxy function block, IO-Link client function blocks (IL_IoLinkCall, IL_IoLinkIdent) and Sercos IO-Link gateway

Profinet-specific IO-Link design

Multiple function blocks already exist for Profinet. They are not in this library, but in the RIL_ProfinetIO.library. However, the proxy concept is not used. The function blocks have a similar functionality. The prefix of the Profinet function blocks is "IL_PNIO" and the prefix of the function blocks of this library is "IL_".

This ensures that as soon as a respective proxy function block is present for Profinet, the same “IL” function blocks can be used from this library.

21.3 IL_SIII_IoLinkProxy

Brief description

The function block IL_SIII_IoLinkProxy implements the proxy function block to communicate to the IO-Link master via Sercos. Proxy is a “substitute” function block. On one hand, the IO-Link is field bus-independent and on the other hand it uses the field bus for communication purposes. This is implemented by the proxy function block. All inputs of the function block are provided with field bus-dependent values. In this case, these values consist of Sercos data such as BusMaster, Sercos address and slot number. All field bus-independent data is transferred to the function block IL_IoLinkCall, page 875. Thus, there is a separation and the IO-Link functionality can be extended to other field buses by simply exchanging the proxy.

⚠ WARNING

The function block may only be instantiated once for each IO-Link master and each Sercos address. If it is instantiated multiple times, incorrect requests and thus incorrect responses result. The, either the function block issues errors or the included data is faulty.

The user is responsible to create only one proxy instance for each Sercos address.

Target system	Library
MLC/MTX	RIL_IoLink
MLD with IndraLogic 2G ≥ 20VRS	RIL_IoLink

Tab. 21-2: Reference table

Interface description

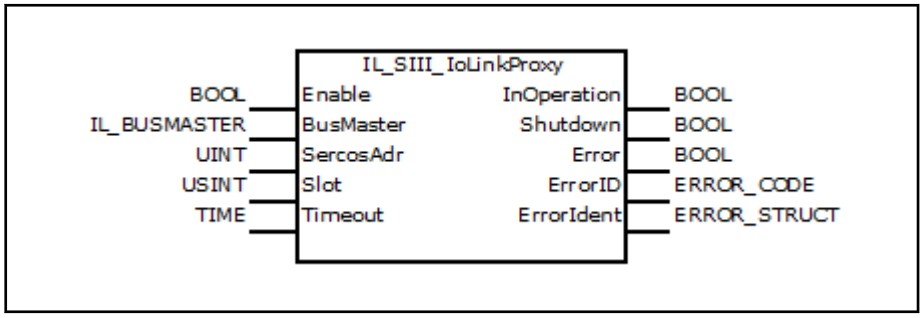


Fig. 21-2: IL_SIII_IoLinkProxy interface

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	BusMaster	IL_BUSMASTER	Sercos bus master selection
	SercosAdr	UINT	Sercos address of the IO-Link master
	Slot	USINT	For modular devices, it is the slot with the IO-Link master
	Timeout	TIME	Maximum time for the function block to receive a response of the IO-Link master
VAR_OUTPUT	InOperation	BOOL	Function block enabled
	Shutdown	BOOL	The function block still executes its required tasks after shutdown

I/O type	Name	Data type	Description
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 874

Tab. 21-3: IL_SIII_IoLinkProxy interface

Name	Data type	Min.	Max.	Default	Effective
Enable	BOOL	-	-	FALSE	Continuous
BusMaster	IL_BUSMAS-TER	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	IL_BUSMAS-TER_0	Rising edge at “Enable”
SercosAdr	UINT	0	511	511	Rising edge at “Enable”
Slot	USINT	0	255	0	Rising edge at “Enable”
Timeout	TIME	0	49d17h2m47s295ms	t#2s	Rising edge at “Enable”

Tab. 21-4: Min./max. and default values of the IL_SIII_IoLinkProxy inputs

Functional description

The function block sets up the communication with an IO-Link master via the Sercos field bus. Set the "Enable" input and the function blocks starts operating. It determines cyclically whether there are present requests of the connected clients, serializes them if required and routes the requests to the IO-Link master. The function block has to be called cyclically. The function block reports errors via its outputs "Error", "ErrorID" and "ErrorIdent". These errors were generally caused by an incorrect parameterization (incorrect bus master, Sercos address, slot). Errors in the IO-Link protocol, errors in the IO-Link communication or IO-Link data are not displayed at this function block, but in the respective client. Thus, the function block generally operates without errors and resulting errors have to be included at the clients. Der "Timeout" has a specific significance. The time entered is used for each client. E.g. if multiple clients are connected, the timeout time is multiplied with the number of clients. A timeout time of 0 means that no timeout is set and that the proxy waits any amount of time.

Internal interfaces are used to communicate between the proxy function block and the connected clients (IL_IoLinkCall). Additionally, data, errors and status information are exchanged in both directions. The communication is only used internally. Thus, the user cannot simplify the function block handling.

An IO-Link master can be provided with multiple "Client Access Points" (CAP). These are parallel options to access the device data. The number of CAPs are manufacturer-specific. The standard specifies only one CAP for Sercos as described in the sub-bus protocol. Multiple requests can simultaneously be sent via this protocol, thus causing again some parallelism. If there are too many requests, the proxy can serialize these requests. The proxy supports up to 8 CAPs. Thus, up to eight client function blocks can connect to one proxy.

pSetLoggerID property

Use this property to change the logger ID. This property is to be used for debugging and may not be used at runtime. Do change this property only if instructed by Bosch Rexroth employees.

Troubleshooting

The function block only generates errors resulting during the Sercos communication with the IO-Link master. Errors within the IO-Link communication do not cause an error at this function block. These errors are generated in the

client function blocks. Resulting errors mostly originate from the subordinate function blocks IL_SIIISvcWrite, IL_SIIISvcRead and IL_SIIICommand.

The most frequent errors are listed in the following:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#00000001	Internal state during the function block processing
STATE_MACHINE_ERROR	16#00000006	16#00000002	Internal error: Invalid state during the function block processing (operation)
STATE_MACHINE_ERROR	16#00000006	16#00000003	Internal error: Invalid state during the function block processing (initialization)
INPUT_RANGE_ERROR	16#00000002	16#00000001	The "BusMaster" input is beyond the permitted range.
INPUT_RANGE_ERROR	16#00000002	16#00000002	The "SercosAdr" input is beyond the permitted range
INPUT_RANGE_ERROR	16#00000002	16#00000007	The "Timeout" input is beyond the permitted range
OTHER_ERROR	16#00000015	16#00000001	The function block was ended due to timeout
DEVICE_ERROR	16#00002E01	16#0000040	Internal error: Semaphore could not be created
DEVICE_ERROR	16#00002E01	16#0000041	Internal error: Semaphore could not be allocated
DEVICE_ERROR	16#00002E01	16#0000042	Internal error: Semaphore could not be shared
DEVICE_ERROR	16#00002E01	16#0000043	Internal error: Semaphore could not be deleted

The user cannot recover internal errors. In this case, restart the function block (set the "Enable" input to "False" and then to "True" again).If the error results multiple times, contact the customer service.

Example code

Refer to the library for example codes to use the function block. There is a folder called "_Examples". The folder includes examples for one single instance connecting to a proxy function block and for multiple client instances connecting to the proxy function block.

21.4 IL_IoLinkCall

Brief description

The function block IL_IoLinkCall implements the client function block for the IO-Link communication to the IO-Link master. Multiple function blocks can be connected to one proxy (IL_IoLinkCall, page 875).

Target system	Library
MLC/MTX	RIL_IoLink
MLD with IndraLogic 2G ≥ 20VRS	RIL_IoLink

Tab. 21-5: Reference table

Interface description

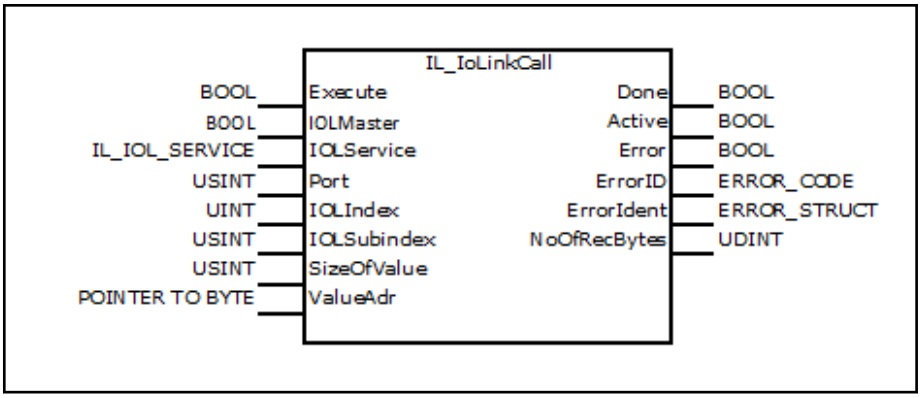


Fig. 21-3: IL_IoLinkCall interface

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Processing of the function block enabled (edge-controlled)
	IOLMaster	BOOL	Selection between IO-Link device and IO-Link master
	IOLService	IL_IOL_SERVICE	Reading or writing data
	Port	USINT	IO-Link port to start reading/writing from (starting at 1)
	IOLIndex	UINT	IO-Link index
	IOLSubindex	USINT	IO-Link subindex
	SizeOfValue	USINT	Data size to be written or available memory (while reading)
	ValueAdr	POINTER TO BYTE	Address of the stored data (write) or of the data to be stored (read)
VAR_OUTPUT	Done	BOOL	The function block completed the processing without errors
	Active	BOOL	The block is still being processed
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 877
	NoOfRecBytes	UDINT	Number of received data in bytes

Tab. 21-6: IL_IoLinkCall interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
IOLMaster	BOOL	FALSE	TRUE	IL_FALSE_0	Rising edge at “Execute”
IOLService	IL_IOL_SERV-ICE	IL_IOL_SERV-ICE.WRITE	IL_IOL_SERV-ICE.READ	IL_IOL_SERV-ICE.READ	Rising edge at “Execute”
Port	USINT	0	63	0	Rising edge at “Execute”
IOLIndex	UINT	0	65535	0	Rising edge at “Execute”
IOLSubindex	USINT	0	255	0	Rising edge at “Execute”

Name	Data type	Min.	Max.	Default	Effective
SizeOfValue	USINT	0	232	0	Continuous
ValueAdr	POINTER TO BYTE	-	-	0	Continuous

Tab. 21-7: Min./max. and default values of the IL_IoLinkCall inputs

Functional description

The function block allows to read and write IO-Link parameters of IO-Link devices and the IO-Link master (selected via "IOLMaster"). Data can be read and written (select via "IOLService"). Other services (Cancel and Idle) are not possible and issue an error. To address IO-Link device parameters, use the inputs "Port", "IOLIndex" and "IOLSubindex". Only the inputs "Port" and "IOLIndex" are used when addressing IL Link masters. "IOLSubindex" has no significance anymore. The inputs "ValueAdr" (pointer to a data area) and "SizeOfValue" (specifies the memory size) are always required. For write access, it specifies the number of bytes to be written. For read access, it specifies the available memory. The function block always operates with a proxy function block. Currently, there is only IL_SIII_IoLinkProxy as proxy function block. The proxy function block instance used has to be specified in the declaration. It cannot be changed at runtime.

Program:

```
fbPProxy  : IL_SIII_IoLinkProxy;      // Declaration of Proxy function block
fbIOLCall : IL_IoLinkCall(fbPProxy);  // Declaration of Client function block
                                            // with initialization of proxy instance
```

The function block only registers at the proxy when the request is active. The function block deregisters from the proxy after the request has been successfully or aborted with error.

pSetLoggerID property

Use this property to change the logger ID. This property is to be used for debugging and may not be used at runtime. Do change this property only if instructed by Bosch Rexroth employees.

Troubleshooting

The function block only generates errors resulting during the IO-Link communication. However, errors are output during the Sercos communication at the proxy function block (IL_SIII_IoLinkProxy). The errors are partially generated in the function block or they are explicitly provided by the IO-Link master and specified in the IO-Link standard (error with Additional1 = 16#00002E03).

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#00000002	Internal state during the function block processing
INPUT_RANGE_ERROR	16#00000002	16#00000005	The "SizeOfValue" input is beyond the permitted range
INPUT_RANGE_ERROR	16#00002E00	16#00000020	The "Proxy" input is invalid
INPUT_RANGE_ERROR	16#00002E00	16#00000021	The "IOLService" input is beyond the permitted range
INPUT_INVALID_ERROR	16#00002E01	16#00000022	Internal error (proxy data address or call interface address invalid)
SYSTEM_ERROR	16#00002E01	16#00000023	Proxy is in error state Request is not processed. Proxy function block has to be restarted first
RESOURCE_ERROR	16#00002E02	16#00000024	IL_IOLinkCall() was called before the proxy "InOperation" was displayed.

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00002E02	16#00000025	Internal error (function block already sent the request. No new request can be sent as long as the old request is still being processed)
RESOURCE_ERROR	16#00002E02	16#00000026	All resources are assigned in the proxy. Currently, no request is possible
RESOURCE_ERROR	16#00002E02	16#00000027	Internal error (function block is deregistered, but has never been registered)
RESOURCE_ERROR	16#00002E02	16#00000028	Internal error (new request is placed, but the function block is not yet registered)
DEVICE_ERROR	16#00002E03	16#00000029	Internal error (command PDU size too small)
DEVICE_ERROR	16#00002E03	16#0000002A	Internal error (port differs in request and response)
DEVICE_ERROR	16#00002E03	16#0000002B	Internal error (command SDU size too small)
DEVICE_ERROR	16#00002E03	16#0000002C	Internal error (command differs in request and response)
DEVICE_ERROR	16#00002E03	16#0000002D	Internal error (index or subindex differ in request and response)
DEVICE_ERROR	16#00002E03	16#0000002E	Internal error (telegram size invalid in response)
DEVICE_ERROR	16#00002E03	16#0000002F	Internal error (data size invalid in response)
DEVICE_ERROR	16#00002E03	16#00000030	Internal error (telegram type invalid in response)
DEVICE_ERROR	16#00002E03	16#00000031	Internal error (sub-bus telegram too small)
DEVICE_ERROR	16#00002E03	16#00000032	Internal error (sub-bus PDU version unknown)
DEVICE_ERROR	16#00002E03	16#00000033	Internal error (sub-bus telegram type is not IO-Link)
DEVICE_ERROR	16#00002E03	16#00000034	Internal error (sub-bus SDU version unknown)
DEVICE_ERROR	16#00002E03	16#00000035	Internal error (sub-bus PDU size and list length do not match)
DEVICE_ERROR	16#00002E03	16#00000036	Proxy was reset (due to error or Enable = False) while requests were still running
DEVICE_ERROR	16#00002E05	16#00008000	Selected port is not available
DEVICE_ERROR	16#00002E05	16#00008001	Selected parameter index ("IOLIndex" input) is not available
DEVICE_ERROR	16#00002E05	16#00008002	Parameter index ("IOLIndex" index) and data do not match within the telegram
DEVICE_ERROR	16#00002E05	16#00008003	Control field is unknown within the request telegram
DEVICE_ERROR	16#00002E05	16#00008004	The selected port cycle time is either not supported or not defined
DEVICE_ERROR	16#00002E05	16#00008005	The configured cycle time is either not supported or not defined
DEVICE_ERROR	16#00002E05	16#00008006	The configured memory mode is either not supported or not configured
DEVICE_ERROR	16#00002E06	16#00008001	The service is not available (e.g. the port is not configured as IO-Link port)
DEVICE_ERROR	16#00002E06	16#00008002	Port assigned by other service

ErrorID	Additional1	Additional2	Description
DEVICE_ERROR	16#00002E06	16#00008003	Timeout if IO-Link device or IO-Link master ports are assigned
DEVICE_ERROR	16#00002E06	16#00008004	Invalid index
DEVICE_ERROR	16#00002E06	16#00008005	Invalid subindex
DEVICE_ERROR	16#00002E06	16#00008007	Port function is not supported
DEVICE_ERROR	16#00002E06	16#00008008	Length invalid
DEVICE_ERROR	16#00002E06	16#00008009	The IO-Link device does not support the protocol used
DEVICE_ERROR	16#00002E06	16#00008010	Values in the “Control” field not defined upon request
DEVICE_ERROR	16#00002E07	16#00008000	General error
DEVICE_ERROR	16#00002E07	16#00008011	Index not available
DEVICE_ERROR	16#00002E07	16#00008012	Subindex not available
DEVICE_ERROR	16#00002E07	16#00008020	Service currently not available
DEVICE_ERROR	16#00002E07	16#00008021	Service currently not available - Locally used
DEVICE_ERROR	16#00002E07	16#00008022	Service currently not available - by IO-Link device
DEVICE_ERROR	16#00002E07	16#00008023	Access not allowed
DEVICE_ERROR	16#00002E07	16#00008030	Parameter beyond the value range
DEVICE_ERROR	16#00002E07	16#00008031	Parameter is greater than the maximum value
DEVICE_ERROR	16#00002E07	16#00008032	Parameter smaller than the minimum value
DEVICE_ERROR	16#00002E07	16#00008033	Parameter length too long
DEVICE_ERROR	16#00002E07	16#00008034	Parameter length too short
DEVICE_ERROR	16#00002E07	16#00008035	Function not available
DEVICE_ERROR	16#00002E07	16#00008036	Function temporarily not available
DEVICE_ERROR	16#00002E07	16#00008040	Invalid parameter set
DEVICE_ERROR	16#00002E07	16#00008041	Inconsistent parameter set
DEVICE_ERROR	16#00002E07	16#00008082	Application not available
DEVICE_ERROR	16#00002E07	16#00008100 .. 16#0000810FF	Manufacturer-specific error

☞ The user cannot recover internal errors. In this case, restart the function block (set the "Enable" input to "False" and then to "True"). If the error is issued multiple times, contact the customer service.

☞ The IO-Link master generates the Additional1 errors using the values 16#00002E05, 16#00002E06 and 16#00002E07. Depending on the manufacturer, more errors can result in addition to the errors listed in Additional2. For its significance, refer to the respective documentation of the IO-Link master.

Example code Refer to the library for an example code to use the function block. There is a folder called "_Examples". The folder includes examples for one single in-

stance connecting to a proxy function block and for multiple client instances connecting to the proxy function block.

21.5 IL_IoLinkErrorText

Brief description

The function IL_IoLinkErrorText converts an error code of the function blocks IL_SIII_IoLinkProxy, page 873, IL_IoLinkCall, page 875 and its derived function blocks into a text.



The function is only intended for engineering purposes. Thus, the output is always in English.

Target system	Library
MLC/MTX	RIL_IoLink
MLD with IndraLogic 2G ≥ 20VRS	RIL_IoLink

Tab. 21-8: Reference table

Interface description

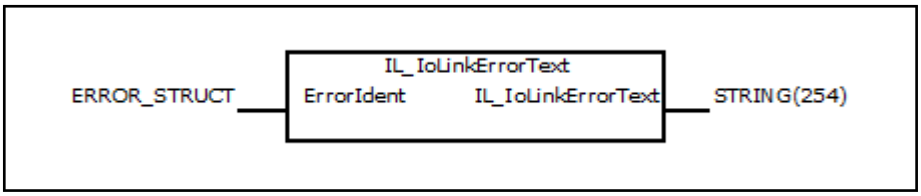


Fig. 21-4: IL_IoLinkErrorText interface

I/O type	Name	Data type	Description
VAR_INPUT	ErrorIdent	ERROR_STRUCT	Error structure of either the IL_SIII_IoLinkProxy or the IL_IoLinkCall or of its derived function blocks with the text to be displayed
VAR_OUTPUT	IL_IoLinkErrorText	STRING(254)	Text in English for the error code

Tab. 21-9: IL_IoLinkErrorText interface

Functional description

The function IL_IoLinkErrorText converts an error code of the function blocks IL_SIII_IoLinkProxy, IL_IoLinkCall and its derived function blocks into text. The error codes are used from the constants IL_IO_LINK_CONSTANTS_ADD1 and IL_IO_LINK_CONSTANTS_ADD1 and provided as text in English. Other languages are not available, as this function was only created for engineering purposes.

21.6 IL_IoLinkIdent

Brief description

This function block queries all mandatory IO-Link parameters required to identify a device. The function block IL_IoLinkIdent implements a client function block for the IO-Link communication to the IO-Link master. Multiple function blocks can be connected to one proxy (IL_SIII_IoLinkProxy, page 873). It is based on the chapter 21.4 "IL_IoLinkCall" on page 875 function block.

The function block retrieves "Direct parameter page 1". The function block then extracts the required information and provides it to the outputs. As the "Direct parameter page 1" is a required parameter, all other parameters are also available. This is not mandatorily the case for other IO-Link parameters.

Target system	Library
MLC/MTX	RIL_IoLink
MLD with IndraLogic 2G ≥ 20VRS	RIL_IoLink

Tab. 21-10: Reference table

Interface description

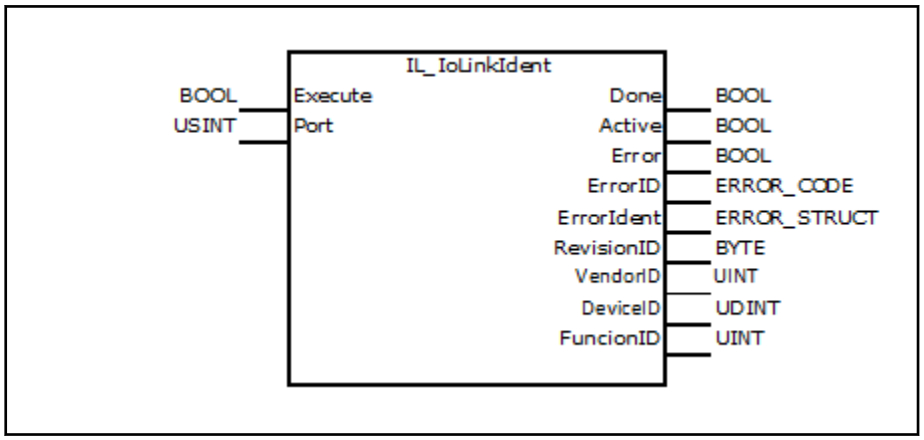


Fig. 21-5: IL_IoLinkIdent interface

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Processing of the function block enabled (edge-controlled)
	Port	USINT	IO-Link port to start reading/writing from (starting at 1)
VAR_OUTPUT	Done	BOOL	The function block completed the processing without errors
	Active	BOOL	The block is still being processed
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Troubleshooting, page 882
	RevisionID	BYTE	Revision of the IO-Link device
	VendorID	UINT	Vendor ID of the IO-Link device
	DeviceID	UDINT	Device ID of the IO-Link device
	FunctionID	UINT	Function ID of the IO-Link device

Tab. 21-11: IL_IoLinkIdent interface

Name	Data type	Min.	Max.	Default	Effective
Execute	BOOL	FALSE	TRUE	FALSE	Continuous
Port	USINT	0	63	0	Rising edge at “Execute”

Tab. 21-12: Min./max. and default values of the IL_IoLinkIdent inputs

Functional description

The function block retrieves the values required to identify an IO-Link device. As it is a required parameter, every device has to support these values. The function block always operates with a proxy function block. Currently, there is only chapter 21.3 "IL_SIII_IoLinkProxy" on page 873 as proxy function block. The proxy function block instance used has to be specified in the declaration. It cannot be changed at runtime.

Program:

```
fbPProxy   : IL_SIII_IoLinkProxy;          // Declaration of Proxy function block
fbIOLIdent : IL_IL_IoLinkIdent(fbPProxy);  // Declaration of Client function block
                                           // with initialization of proxy instance
```

The function block only registers at the proxy when the request is active. The function block deregisters from the proxy after the request was successfully or aborted with error.

pSetLoggerID property Use this property to change the logger ID. This property is to be used for debugging and may not be used at runtime. Do change this property only if instructed by Bosch Rexroth employees.

Troubleshooting The function block only generates errors resulting during the IO-Link communication. However, errors during the Sercos communication are output at the proxy function block (IL_SIII_IoLinkProxy). The errors are partially generated in the function block or explicitly provided by the IO-Link master and specified in the IO-Link standard (error with Additional1 = 16#00002E03). As the function block is subordinately based on the IL_IoLinkCall, the same errors are generated (refer to "Troubleshooting" on page 877).

Example code Refer to the library for an example code to use the function block. There is a folder called "_Examples". The folder includes examples for one single instance connecting to a proxy function block and for multiple client instances connecting to the proxy function block.

21.7 RexrothHydraulicValve

21.7.1 IL_IoLinkDirectValveControlType01

Brief description The function block IL_IoLinkDirectValveControlType01 extends the function block IL_EnableInOpBase and it is used to control a Bosch Rexroth I/O link valve.

Target system	Library
MLC/MTX	RIL_IoLink
MLD with IndraLogic 2G for 20VRS and above	RIL_IoLink

Tab. 21-13: Reference table

Interface description

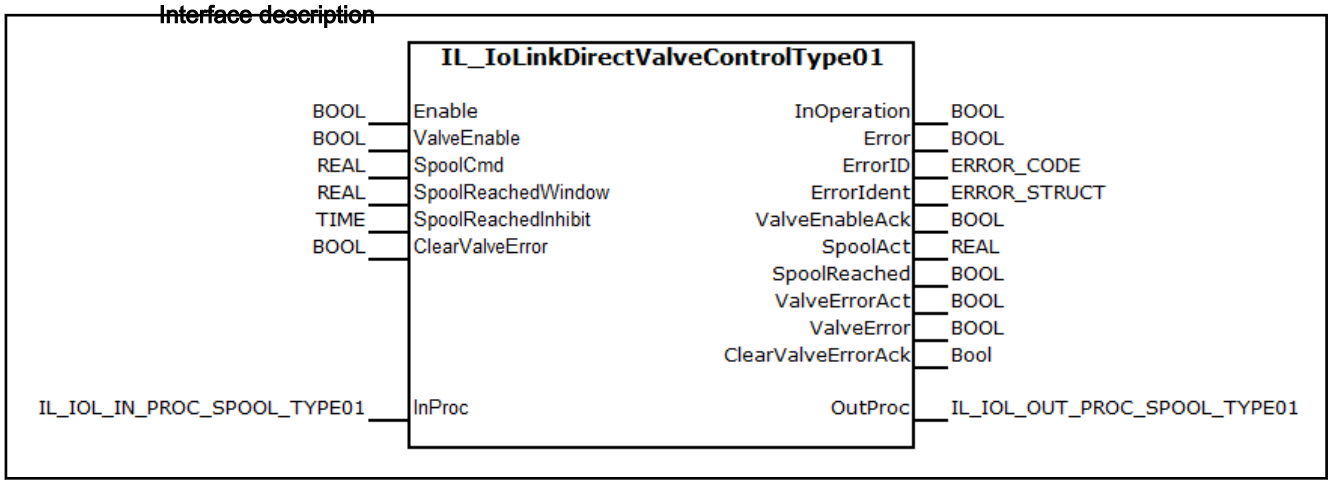


Fig. 21-6: IL_IoLinkDirectValveControlType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled)
	ValveEnable	BOOL	Enabling the valve
	SpoolCmd	REAL	Valve command value. Unit in %. The workspace is +/-100%
	SpoolReached-Window	REAL	Target range around the command value. Unit in %
	SpoolReache-dInhibit	TIME	Time in the target range before “SpoolReached” is set
	ClearValveEr-ror	BOOL	Clearing valve error
	InProc	IL_IOL_IN_PROC_SPOOL_TYPE01	Applying process input data %I IL_IOL_IN_PROC_SPOOL_TYPE01, page 886
VAR_OUTPUT	InOperation	BOOL	Function block enabled
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Diagnostic description in case of error
	ErrorIdent	ERROR_STRUCT	Detailed diagnostics Error description, page 885
	ValveEna-bleAck	BOOL	Valve enabled acknowledged
	SpoolAct	REAL	Actual valve value. Unit in %
	SpoolReached	BOOL	Valve command value = Actual valve value +/- Spool-ReachedWindow
	ValveErrorAct	BOOL	Valve error present. More information on the error cause can be retrieved acyclically from the I/O link device. For function blocks for the acyclic communication, refer to the library on the field bus used. E.g. for Profinet, refer to the library RIL_ProfinetIO, folder IO-Link.
	ValveError	BOOL	A valve error was identified (clear after the last valve error)
	ClearValveEr-rorAck	BOOL	Clearing valve error acknowledged
	OutProc	IL_IOL_OUT_PROC_SPOOL_TYPE01	Applying process output data %Q IL_IOL_OUT_PROC_SPOOL_TYPE01, page 885

Tab. 21-14: IL_IoLinkDirectValveControlType01 interface

Min./max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
ValveEnable	BOOL			FALSE	Continuous
SpoolCmd	REAL	-110	+110	0%	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
SpoolReachedWind-ow	REAL	0	50	3%	Continuous
SpoolReachedInhibit	TIME			TIME#4 ms	Continuous
ClearValveError	BOOL			FALSE	Continuous
InProc	IL_IOL_IN_PROC_SPOOL_TYPE01			0	Continuous

Functional description

The function block sends and receives either from the valve IL_IOL_IN_PROC_SPOOL_TYPE01 or IL_IOL_OUT_PROC_SPOOL_TYPE01. This includes a 14-bit command or actual value of the valve. This value is then converted into percent. In addition, error messages of the valve, the command to enable the valve and an acknowledgement of this enabling is transferred.

The input "SpoolCmd" specifies the valve command value. The value is cyclically transferred to the valve via the output "OutProc.PDO".

The input "SpoolReachedWindow" specifies a percentage window around a command value. If the actual value is within this range, the command value is considered to be reached. The percentage value refers to +100%.

The input "SpoolReachedInhibit" specifies the minimum time that the actual valve value has to be in the specified range around the command value (see "SpoolReachedWindow") before the output “SpoolReached” becomes active. If the actual valve value leaves the range around the command value, the output "SpoolReached" is disabled immediately without waiting for the time set.

In the following figure, 5 % is set for "SpoolReachedWindow" and a time of 4 ms for "SpoolReachedInhibit".

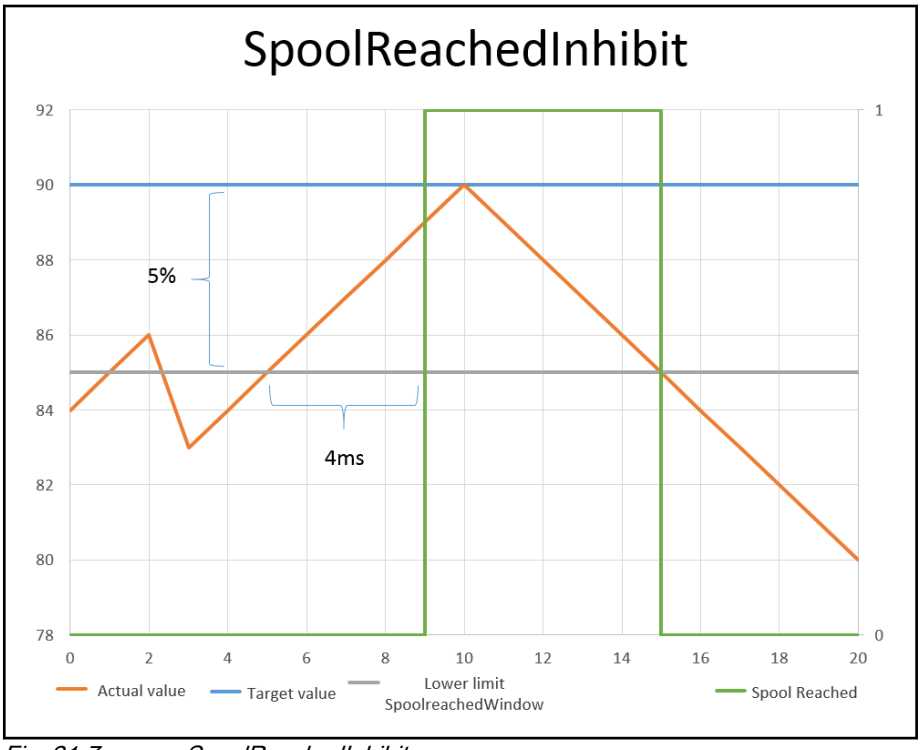


Fig. 21-7: SpoolReachedInhibit

- The output "SpoolReached" is enabled when the actual valve value reaches the valve command value.
- The output "SpoolAct" returns the actual valve value. The value is cyclically read by the valve via the "InProc.PDI".

Configuring cyclic data; example Profinet:

To establish the cyclic communication between the function block and the valve, copy the parameters to the respective input and output variables. The cyclically transferred parameters are configured in the Project Explorer below the PROFINET_IO_Controller node. However, the respective IO Link gateway has to be added below the selected node. If the gateway is connected and the device description file is installed, right-click on the node and "Scan devices" or "Scan bus configuration" (install the device description file via **Tools ▸ Device database**).

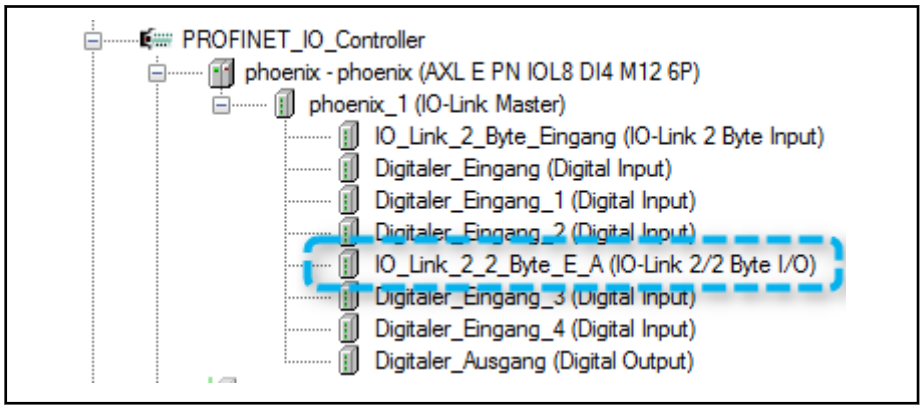


Fig. 21-8: Configuring cyclic data (example Profinet IO <> IO Link gateway)

Then, the IO Link devices have to be added with the respective data width. The outputs "InProc.PDI" and "OutProc.PDO" can be transferred to these devices in the "I/O mapping".

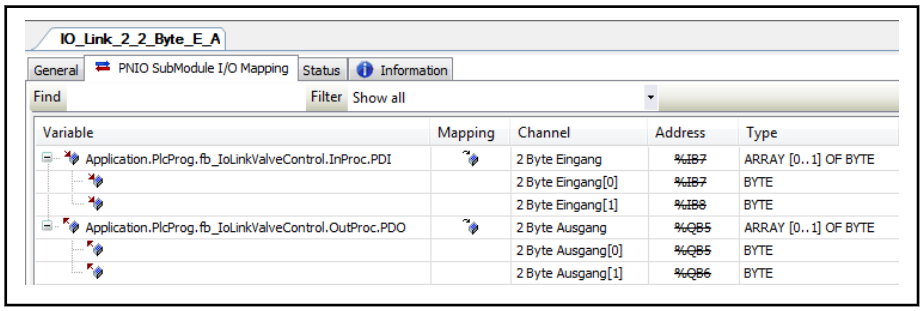


Fig. 21-9: Mapping input and output variables (example IO <> IO Link gateway)

Error description

Specific error codes of the function block:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00000002	16#00000001	The input "SpoolCmd" is beyond the valid range
INPUT_INVALID_ERROR	16#00000002	16#00000002	The input "SpoolReachedWindow" is beyond the valid range

To clear an error, resolve the error cause and set the “Enable” input to FALSE and then to TRUE again.

21.7.2 IL_IOL_OUT_PROC_SPOOL_TYPE01

Brief description

The data structure IL_IOL_OUT_PROC_SPOOL_TYPE01 contains the process output data of the function block IL_IoLinkDirectValveControlType01 on page 882.

Structural element	Type	Default value	Description
PDO	WORD		Output variable

Tab. 21-15: Elements of the IL_IOL_OUT_PROC_SPOOL_TYPE01 structure

Output process data PDO - Total bit length = 16

Name	Description	Data type	Bit offset	Bit length	Value range
Valve command value	Current valve command value	IntegerT	2	14	-110% = 0x8000 -100% = 0x8BB0 -0.0134% = 0xFFFC 0% = 0 +0.0134% = 0x0004 +100% = 0x7450 +110% = 0x7FFC
Not used			1	1	
Enabling	High -> Output stage active Low -> Output stage not active	BooleanT	0	1	(False) inactive (True) active

Tab. 21-16: Structure of the output process data PDO

21.7.3 IL_IOL_IN_PROC_SPOOL_TYPE01

Brief description The data structure IL_IOL_IN_PROC_SPOOL_TYPE01 contains the process input data of the function block IL_IoLinkDirectValveControlType01 on page 882.

Structural element	Type	Default value	Description
PDI	WORD		Input variable

Tab. 21-17: Elements of the IL_IOL_IN_PROC_SPOOL_TYPE01 structure

Input process data PDI - Total bit length = 16

Name	Description	Data type	Bit offset	Bit length	Value range
Actual valve value	Current actual valve value	IntegerT	2	14	-110% = 0x8000 -100% = 0x8BB0 -0.0134% = 0xFFFC 0% = 0 +0.0134% = 0x0004 +100% = 0x7450 +110% = 0x7FFC
Valve OK	High -> No error Low -> Error	BooleanT	1	1	(False) inactive (True) active
Valve On	High -> Output stage active Low -> Output stage not active	BooleanT	0	1	(False) inactive (True) active

Tab. 21-18: Structure of the input process data PDI

22 Modbus RTU - Libraries

22.1 IoDrvModbusComPort

Brief description

Library: `IoDrvModbus`

This function block is generated for each serial Modbus master in the device tree.

Interface description

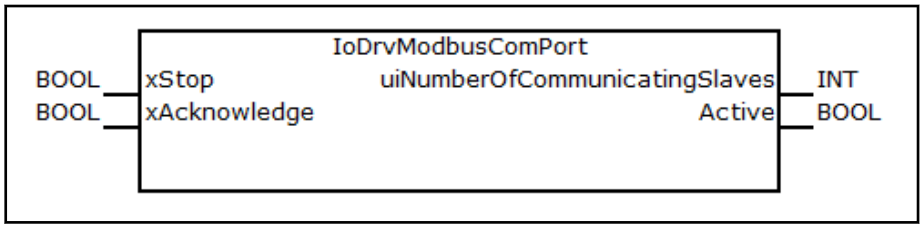


Fig. 22-1: IoDrvModbusComPort function block

I/O type	Name	Data type	Initial value	Description
VAR_INPUT	xStop	BOOL		If `TRUE`, each new request to the slaves is stopped. It is continued at `FALSE`
	xAcknowledge	BOOL		Opens and closes the COM port immediately at a rising edge
VAR_OUTPUT	uiNumberOfCommunicating-Slaves	INT	0	Processing completed without error, output data valid
	Active	BOOL	`FALSE`	Processing active, output data invalid

Tab. 22-1: IoDrvModbusComPort interface

22.2 ModbusSlaveComPort

Brief description

Library: `IoDrvModbus`

This function block is generated for each serial Modbus slave in the device tree. This function block represents the logical connection to a single Modbus serial slave. The channels defined for this device are processed in the background. The inputs/outputs of this function block provide status information and allow to acknowledge errors for example (to continue processing).

Interface description

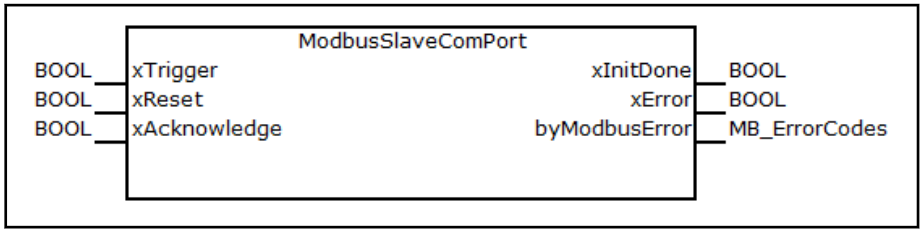


Fig. 22-2: ModbusSlaveComPort

I/O type	Name	Data type	Initial value	Description
VAR_INPUT	xTrigger	BOOL	`FALSE`	Triggers the sending of all configured Modbus channels at a rising edge
	xReset	BOOL	`FALSE`	Restarts the communication and resets `xError` and `byModbusError`

I/O type	Name	Data type	Initial value	Description
	xAcknowledge	BOOL	FALSE	Restarts the communication and does not reset xError and byModbusError
VAR_OUTPUT	xInitDone	BOOL	FALSE	All initialization commands were reset
	xError	BOOL	FALSE	Is TRUE if any error occurs. The communication to this slave is interrupted
	byModbusError	MB_ErrorCodes	0	Current error defined in the enumeration MB_ErrorCodes

Tab. 22-2: ModbusSlaveComPort

22.3 IoDrvModbusTCP

Brief description

Library: IoDrvModbusTCP

This function block is generated for each Modbus master TCP in the device tree.

Interface description

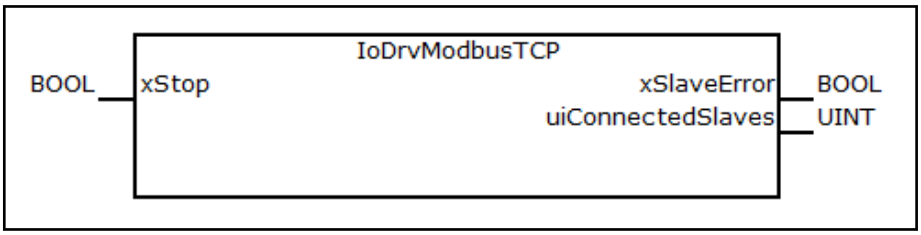


Fig. 22-3: IoDrvModbusTCP function block

I/O type	Name	Data type	Initial value	Description
VAR_INPUT	xStop	BOOL	FALSE	Stops all TCP/IP communication
VAR_OUTPUT	xSlaveError	BOOL	FALSE	Signals that there is a slave function block with an error
	uiConnectedSlaves	UINT	0	Number of slaves with a TCP/IP connection

Tab. 22-3: IoDrvModbusTCP interface

22.4 ModbusTCPSlave

Brief description

Library: IoDrvModbusTCP

This function block is generated for each Modbus slave TCP in the device tree. This function block represents a TCP/IP connection to a single ModbusTCP slave. The channels defined for this device are processed in the background. The inputs/outputs of this function block provide status information and allow to acknowledge errors for example (to continue processing). The communication can be started or stopped via the property “xEnable”.

Interface description

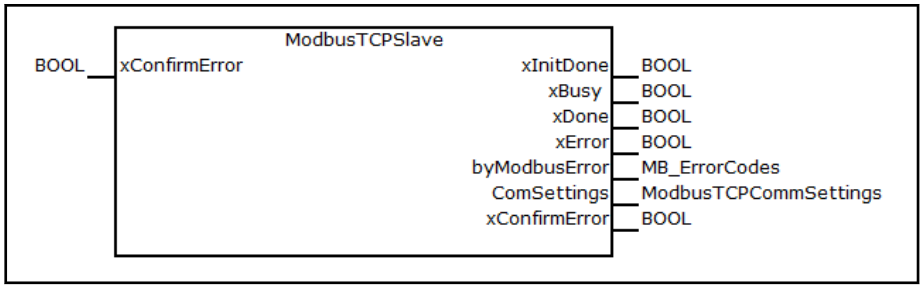


Fig. 22-4: ModbusTCPSlave function block

I/O type	Name	Data type	Initial value	Description
VAR_INPUT	xConfirmError	BOOL	FALSE	The process is continued at a rising edge
VAR_OUTPUT	xInitDone	BOOL	FALSE	Displays the number of communication slaves
	xBusy	BOOL	FALSE	Is TRUE as long as a request is being processed
	xDone	BOOL	FALSE	Is TRUE if a request was completed successfully
	xError	BOOL	FALSE	Is TRUE if a request was completed with error
	byModbusError	MB_ErrorCodes		Specifies the current error as defined in the enumeration MB_ErrorCodes
	ComSettings	ModbusTCP-CommSettings		Currently configured communication settings
	xConfirmError	BOOL	FALSE	The process is continued at a rising edge

Tab. 22-4: ModbusTCPSlave

"UpdateCommunicationSettings" method

The communication parameters can only be updated if the slave is on "Stop". The slave is stopped and started via the property `xEnable`.

I/O type	Name	Data type	Initial value	Description
VAR_INPUT	ipAdress	ARRAY [0..3] OF BYTE		IP address of the slave
	uiPort	UINT		Port of the slave
VAR_OUTPUT	UpdateCommunicationSettings	UDINT		Causes an error (return value != 0) if there is a TCP connection

22.5 IoDrvModbusSerialSlave

Brief description

Library: `IoDrvModbusSerialSlave`

This function block is generated for each serial Modbus slave device in the device tree.

Interface description

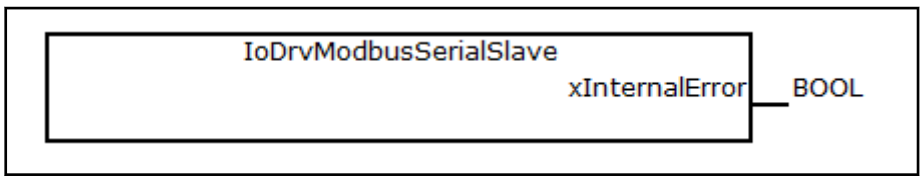


Fig. 22-5: IoDrvModbusSerialSlave function block

I/O type	Name	Data type	Initial value	Description
VAR_OUTPUT	xInternalError	BOOL	FALSE	Non-recoverable internal error (e.g. if mem-aloc fails or if no license is available)

Tab. 22-5: IoDrvModbusSerialSlave interface

22.6 IoDrvModbusTCPSlave

Brief description

Library: IoDrvModbusTCPSlave

This function block is generated for each Modbus TCP slave device in the device tree.

Interface description

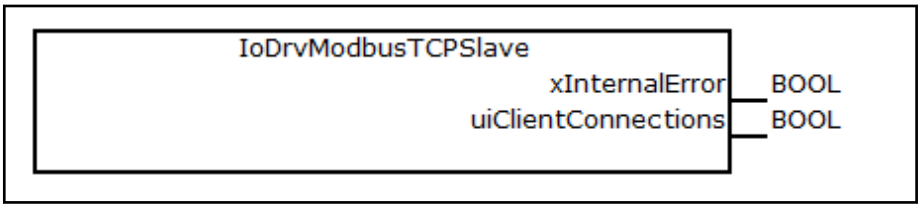


Fig. 22-6: IoDrvModbusTCPSlave function block

I/O type	Name	Data type	Initial value	Description
VAR_OUTPUT	xInternalError	BOOL	FALSE	Non-recoverable internal error (e.g. if mem-aloc fails or if no license is available)
	uiClientConnections	BOOL	FALSE	Number of currently established client connections (TCP/IP)

Tab. 22-6: IoDrvModbusTCPSlave interface

22.7 ModbusServer

Brief description

Library: IoDrvModbusTCPSlave

The function block processes requests of the Modbus TCP. Thus, up to 10 parallel client connections can be processed. Modbus request to data areas mapping input and holding registers.

The following Modbus function codes are supported:

- 0x01 Read Coils
- 0x02 Read Discrete Inputs
- 0x03 Read Holding Registers
- 0x04 Read Input Registers
- 0x05 Write Single Coil
- 0x06 Write Single Register
- 0x0F Write Multiple Coils
- 0x10 Write Multiple Registers
- 0x17 Read/Write Multiple Registers

The ModbusServer function blocks uses the parameters configured in the "Device editor" tab of the Modbus TCP slave.

Interface description

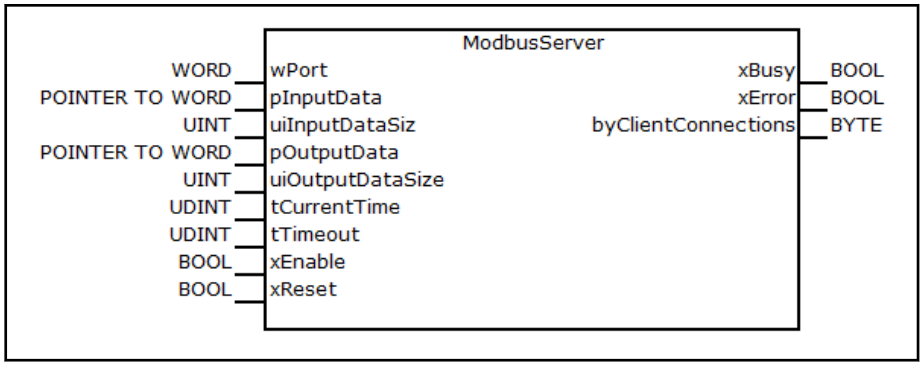


Fig. 22-7: ModbusServer function block

I/O type	Name	Data type	Initial value	Description
VAR_INPUT	wPort	WORD	`502`	Server port (TCP/IP protocol stack) 502 is Modbus standard
	pInputData	POINTER TO WORD		Pointer to WORD pointing to the image of the Modbus input register
	uiInputDataSiz	UINT	`1024`	Length of the arrays of input data (WORDs per register): Default: 1024
	pOutputData	POINTER TO WORD		Pointer to WORD pointing to the image of the Modbus holding register
	uiOutputDataSize	UINT		Length of the arrays of output data (WORDs per register)
	tCurrentTime	UDINT		Current PLC time in ms. If a timeout was configured, it has to be updated before executing the function block
	tTimeout	UDINT		Timeout in ms. If it is not 0 and no Modbus write error occurred up to the timeout, output data is set to 0
	xEnable	BOOL	`FALSE`	Flag to start the request processing
	xReset	BOOL	`FALSE`	Restart of the normal communication with the slaves and reset of the status words and the error bit at a rising edge
VAR_OUTPUT	xBusy	BOOL	`FALSE`	Is TRUE as long as the request is being processed
	xError	BOOL	`FALSE`	Information on the errors found
	byClientConnections	BYTE	0	Number of connections

Tab. 22-7: ModbusServer

22.8 ModbusRequest

Brief description

Library: `IoDrvModbusTCP`

In addition to the Modbus commands predefined in the tab "Modbus slave channel", this function block can be used to dynamically execute further commands in the application. There is no I/O mapping for these programmatically

generated commands, but input/output data areas as `ARRAY OF BYTE` or `ARRAY OF WORD`.

Interface description

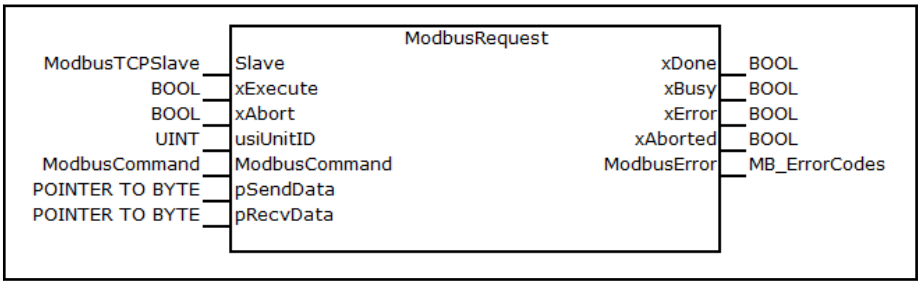


Fig. 22-8: ModbusRequest function block

I/O type	Name	Data type	Initial value	Description
VAR_INPUT	Slave	ModbusTCP-Slave		
	xExecute	BOOL		Rising edge: The action starts Falling edge: The outputs are reset at a falling edge before the function block has completed the action. The outputs respond normally and are only reset if the action is completed or aborted (`xAbort`) or if an error occurs. In this case, the respective output values (`xDone, xError, xError`) are present at the outputs for exactly one cycle
	xAbort	BOOL		`TRUE`: The action is stopped immediately and all outputs are reset to their initialization value
	usiUnitID	UINT	`15#FF`	The unit ID is inly used to route (gateway) to a Modbus serial bus. For TCP/IP, the Modbus server is addressed via its IP address. Modbus unit ID is not used in this case (value `0xFF` has to be used). In case of a serial connection, the Modbus devices are addressed with unit IDs between 1 and 247. 0 is used for broadcast addresses
	ModbusCommand	Modbus-Command		The executed Modbus command. Function blocks for bit access and word access are supported
	pSendData	POINTER TO BYTE		Pointer to a buffer with data that is sent to the slave. For the register access, the buffer type is `Array[0..ModbusCommand.uiWriteLen - 1] OF WORD`. For the bit access, the buffer type is `ARRAY[0..ModbusCommand.uiWriteLen / 8] OF BYTE`

I/O type	Name	Data type	Initial value	Description
	pRecvData	POINTER TO BYTE		Pointer to a buffer in which data received by the slave was written. For the register access, the buffer type is `ARRAY[0..ModbusCommand.uiReadLen - 1] OF WORD`. For the bit access, the buffer type is `ARRAY[0..ModbusCommand.uiReadLen / 8] OF BYTE`
VAR_OUTPUT	xDone	BOOL	`FALSE`	Is `TRUE` if a request was completed successfully
	xBusy	BOOL	`FALSE`	Is `TRUE` as long as the request is being processed
	xError	BOOL	`FALSE`	Is `TRUE` if a request was completed with error
	xAborted	BOOL	`FALSE`	Is `TRUE` if the request was aborted by the user
	ModbusError	MB_ErrorCodes		Current error defined in the enumeration `MB_ErrorCodes`

Tab. 22-8: ModbusRequest

23 RIH_IndraControl_S20.library

23.1 Introduction and overview

The library **RIH_IndraControl_S20** describes the access options of the IEC program to the IndraControl S20 system using function blocks.

The IndraControl S20 system is used for IndraControl XM controls, IndraControl S20 bus couplers and IndraControl S20 modules.

Available functions for IndraControl XM control + IndraControl S20 bus couplers and IndraControl S20 modules:

- Access to PDI objects
- Diagnostics
- Type plate
- Normalizing input/output data
- Configuration

IndraControl S20 station: Example with bus coupler

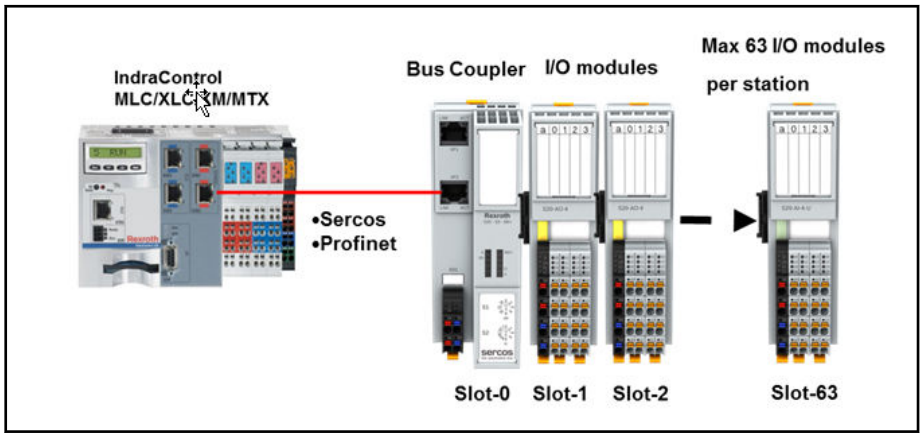


Fig. 23-1: IndraControl S20 station: Example with bus coupler

The library can be used for the following categories/modules:

Category/module	Function block	Description
Diagnostics		
IndraControl XM	IH_S20IOState, page 901	Outputs the state of the local S20 I/O
IndraControl XM	IH_S20IOStateDetails, page 902	Outputs a detailed state of the local S20 I/O to identify the error location and cause
IndraControl S20 modules at an IndraControl XM	IH_S20IOModuleStateDetails, page 900	Outputs a detailed state of the local S20 module connected to an IndraControl S20 to identify the error location and cause
IndraControl S20 bus couplers and modules	IH_S20ModuleDiagnostics	Outputs the diagnostics of an S20 module connected to an IndraControl S20 bus coupler (PDI object 0x18 is read)
I&M (Identification and Maintenance) (IM Data)		
IndraControl S20 bus couplers and modules	IH_S20ModuleIdentDetails, page 903	Reading type plate and writing from "Tag Location"
Acyclic communication (Acyclic Communication)		

Category/module	Function block	Description
IndraControl S20 bus couplers and modules	IH_S20ComInterface, page 906	Read-only and write access to the PDI objects of the IndraControl S20 modules. For an overview on the supported PDI objects, refer to the data sheets of the module. Used as basis for other function blocks.
IndraControl XM with IndraControl S20 bus couplers and modules	IH_S20ComConfiguration, page 904	Generates a communication handle for the acyclic *Type02 function blocks. A new instance is required for each bus master (IndraControl XM or IndraControl S20 bus coupler), not for each IndraControl S20 module
IndraControl XM with IndraControl S20 bus couplers and modules	IH_S20ComInterfaceType02, page 905	Read-only and write access to the PDI objects of the IndraControl S20 modules. For an overview on the supported PDI objects, refer to the data sheets of the module. Used as basis for other function blocks
IndraControl XM with IndraControl S20 modules	IH_S20IOPDIRead	Read-only access to the PDI objects of the IndraControl S20 modules. For an overview on the supported PDI objects, refer to the data sheets of the module. Used as basis for other function blocks
IndraControl XM with IndraControl S20 modules	IH_S20IOPDIWrite	Write access to the PDI objects of the IndraControl S20 modules. For an overview on the supported PDI objects, refer to the data sheets of the module. Used as basis for other function blocks
Analog input modules (AnalogIO)		
S20-AI-4-I	IH_S20AI4IU, page 912 IH_S20AI4IUType02	Normalizing values *Type02 including configuration
S20-AI-4-U	IH_S20AI4IU, page 912 IH_S20AI4IUType02	Normalizing values *Type02 including configuration
S20-AI-8	IH_S20AI8, page 916 IH_S20AI8Type02	Normalizing values *Type02 including configuration
S20-AI-4-RTD	IH_S20AI4RTD, page 918 IH_S20AI4RTDType02	Normalizing values *Type02 including configuration
S20-AI-4-UTH	IH_S20AI4UTH, page 920 IH_S20AI4UTHType02	Normalizing values *Type02 including configuration
S20-AI-8-RTD	IH_S20AI8RTD, page 922 IH_S20AI8RTDType02	Normalizing values *Type02 including configuration
S20-AI-8-UTH	IH_S20AI8UTH, page 924 IH_S20AI8UTHType02	Normalizing values *Type02 including configuration
Analog output modules (AnalogIO)		

Category/module	Function block	Description
S20-AO-4	IH_S20AO4, page 928 IH_S20AO4Type02	Normalizing values *Type02 including configuration
S20-AO-8	IH_S20AO8, page 926 IH_S20AO8Type02	Normalizing values *Type02 including configuration
Analog input/output modules (AnalogIO)		
S20-AIAO-2		Being prepared
Technology modules (TechnologyIO)		
S20-AI6-AO2-SSI2	IH_S20Ai6Ao2Ssi2	Normalizing values
S20-CNT-INC-2/2	IH_S20CntInc, page 935	Normalizing values
S20-SSI-AO-1/1	IH_S20SsiAo1, page 931	Normalizing values
S20-RS-UNI	IH_S20RSCom, page 940 IH_S20RSComType02	Sending and receiving serial data *Type02 including configuration

Tab. 23-1: RIH_IndraControl_S20 library function blocks

The library supports the following functions for the function blocks *Type02:

Functions for analog modules	Description
IH_S20AIConfig	The function supports when generating configuration data for function blocks with analog inputs
IH_S20AOConfig	The function supports when generating configuration data for function blocks with analog outputs
IH_S20TEMPConfig	The function supports when generating configuration data for function blocks with RTD
IH_S20UTHConfig	The function supports when generating configuration data for function blocks with UTH

Tab. 23-2: RIH_IndraControl_S20 library functions

S20 modules can be accessed acyclically via PDI objects using the library. For the PDI objects, refer to the respective data sheet of the S20 module. An S20 module is addressed via the "Slot" input.

The library supports Sercos, Profinet, Profibus, Ethernet/IP, Ethernet (Modbus TCP) and S20 I/O local (currently tested with XM 20-IO local, Sercos, Profinet and Profibus).

- The IndraControl S20 system is configured with IndraWorks. The function blocks **without** the "suffix *Type02" read the configuration of the IndraControl S20 modules at an IndraControl S20 bus coupler.
- The function blocks **with** the "suffix *Type02" (as of 14V22) do additionally run with the IndraControl XM at the local bus and allow to write the configuration from the PLC to the IndraControl S20 modules. It is written on the PDI object 0x80 ParamTable. Before writing, the configuration is read to ensure that there are only required writing operations.

  Known problem:

  At the moment, a field bus startup continuously overwrites the configuration with the configuration from IndraWorks.

  Field bus startup means:

  – Downloading PLC program
  – Disconnecting and the connecting Profibus/Profinet cables at the bus coupler
  – Switching Sercos phase

  Workaround:

  Removing parameter data from the device description files. Then, only function blocks are used for the configuration. The process data mapping is still performed in IndraWorks.

Documentation

For the documentation on each function block, go to the "Documentation" tab in the IndraWorks library manager when selecting the function block.

The "Documentation" tab includes the following information:

- Special features on the function blocks
- Structures of the function blocks
- Enumerations of the function blocks

The code examples and error codes are listed in the following.

Code example provide support. For more examples, refer to the "_CodeExample" folder

23.2 Error codes

Troubleshooting

The function blocks of the library uses the following error tables:

- **S20_IO_TABLE** (ERROR_TABLE=16#0210)
- **SERCOS_TABLE** (ERROR_TABLE=16#0010)
- **PROFINETIO_TABLE** (ERROR_TABLE=16#0200)
- **ETHERNET_IP_TABLE** (ERROR_TABLE=16#0151)
- **ETHERNET_TABLE** (ERROR_TABLE=16#0150)
- **F_RELATED_TABLE** (ERROR_TABLE=16#0170)

23.3 Diagnostics

23.3.1 IH_S20IOModuleStateDetails

Example program

The diagnostics of the S20 modules locally connected to the XM control is shown in the example.

Each S20 module can read up to one diagnostics that is the last one with the highest priority.

The number of the connected S20 modules can correspond to the structure at the "DiagInfoAdr" input with regard to size. This allows to read the diagnostics from each connected S20 module. In practice, all modules rarely report an error at the same time. Thus, the size of the structure can be smaller than the number of the connected modules.

Recommendation:

- For S20 modules ≤10, the size corresponds to the number of S20 modules
- For S20 modules ≥10, the size of the structure is between 10 and 20

TRUE at the "Execute" input starts retrieving the diagnostics.

This works only with an S20 control based on a local bus, e.g. IndraControl XM.

Declaration:

Program:

```
PROGRAM ProgIH_S20ModuleDiagnostics
VAR
    fbIH_S20IOModuleStateDetails    : IH_S20IOModuleStateDetails;    // Function block instance
    bExecute    : BOOL;             // TRUE: Function is enabled
    bDone       : BOOL;        // TRUE: Function Block has completed its task
    bActive       : BOOL;        // TRUE: Function Block is working on its actual task
    bError      : BOOL;        // TRUE: Indicates an error
    ErrorID     : ERROR_CODE;    // Class of error
    ErrorIdent  : ERROR_STRUCT;    // Detailed information about error
    arDiagInfoAdr  : ARRAY[1..20] OF IH_S20_IO_DIAGNOSTICS; // Address of Diagnostics information array.
                    // Use ADR () operator. Contains Diagnostic information of the modules.
    lwListOfDiagModules : LWORD; // Bitwise representation of module slots, where diagnostic messages exist
    udiActiveDiagCount  : UDINT; // Number of diagnostic messages currently active/exist
    udiNoOfDiagInfo        : UDINT; // Number of diagnostic messages read and processed
    udiMoreDiagInfoCount: UDINT; // Number of diagnostic messages still exist but not yet processed
END_VAR
```

Program:

Program:

```
fbIH_S20IOModuleStateDetails(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    DiagInfoAdr:= ADR(arDiagInfoAdr),
    SizeOfDiagInfo:= SIZEOF(arDiagInfoAdr),
    ListOfDiagModules=> lwListOfDiagModules,
    ActiveDiagCount=> udiActiveDiagCount,
    NoOfDiagInfo=> udiNoOfDiagInfo,
    MoreDiagInfoCount=> udiMoreDiagInfoCount);
```

23.3.2 IH_S20IOState

Example program

The example shows the reading of a status WORD to diagnose the S20 I/O master of an XM control.

TRUE at the "Enable" input starts retrieving the diagnostics.

☞ This works only with an S20 control based on a local bus, e.g. IndraControl XM.

Variable declaration:

Program:

```
PROGRAM ProgIH_S20IOState
VAR
    fbIH_S20IOState: IH_S20IOState; // Function block instance
 bEnable     : BOOL;  // TRUE: Function is enabled
    bDone       : BOOL;  // TRUE: Function Block has completed its task
    bActive     : BOOL;  // TRUE: Function Block is working on its actual task
    bError      : BOOL;  // TRUE: Indicates an error
    ErrorID     : ERROR_CODE;  // Class of error
    ErrorIdent  : ERROR_STRUCT;  // Detailed information about error
 wState      : WORD;  // S20 I/O bus diagnostic status
END_VAR
```

Program:

Program:

```
fbIH_S20IOState(
    Enable:= bEnable,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    State=> wState);
```

23.3.3 IH_S20IOStateDetails

Example program

The example shows the reading of a detailed analysis of the S20 I/O of an XM control.

TRUE at the "Execute" input starts retrieving the diagnostics.

☞ This works only with an S20 control based on a local bus, e.g. IndraControl XM.

Variable declaration:

Program:

```
PROGRAM ProgIH_S20IOStateDetails
VAR
    fbIH_S20IOStateDetails : IH_S20IOStateDetails; // Function block instance
 bExecute : BOOL;// TRUE: Function is enabled
    bDone : BOOL;// TRUE: Function Block has completed its task
    bActive : BOOL;// TRUE: Function Block is working on its actual task
    bError : BOOL;// TRUE: Indicates an error
    ErrorID    : ERROR_CODE;// Class of error
    ErrorIdent    : ERROR_STRUCT;    // Detailed information about error
 stStateDetails : IH_S20_MASTER_STATE; // S20 I/O bus diagnostics data.
                 // Refer structure: IH_S20_MASTER_STATE
    stDiagGlobalStatus : IH_S20_DIAG_GLOBAL_STATUS;
                      //S20 I/O system different error count information.
                      //Refer structure: IH_S20_DIAG_GLOBAL_STATUS
END_VAR
```

Program:

Program:

```
fbIH_S20IOStateDetails(
```

```
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    StateDetails=> stStateDetails,
    DiagGlobalStatus=> stDiagGlobalStatus);
```

23.4 I&M (Identification and Maintenance) (IM Data)

23.4.1 IH_S20ModuleIdentDetails

Example program

The example shows the reading of a detailed analysis of the S20 I/O of an XM control.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address/name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "Execute" input starts retrieving the diagnostics to identify (device type plate) of the S20-AI-4-I.

Declaration:

Program:

```
PROGRAM ProgIH_S20ModuleIdentDetails
VAR
    fbIH_S20ModuleIdentDetails    : IH_S20ModuleIdentDetails; // Function block instance
    bExecute   : BOOL; // Function block processing enabled( TRUE - once, edge-controlled )
    bDone   : BOOL; // TRUE: Function Block has completed its task
    bActive   : BOOL; // TRUE: Function Block is working on its actual task
    bError   : BOOL; // TRUE: Indicates an error
    ErrorID   : ERROR_CODE; // Class of error
    ErrorIdent   : ERROR_STRUCT; // Detailed information about error
    enBusType   : IH_S20_BUSTYPE; // Bus type selection
    stBusConfig   : IH_S20_BUS_CONFIG; // Bus configuration parameters
    uiSlot   : UINT := 1; // Module slot address. Bus coupler : 0,
                       // 1st Module after coupler:1 ... 63rd Module after coupler: 63
    enIdentMode   : IH_S20_IDENTMODE; // Enumeration for read/write of module Identification objects
    stManufacturerDetails  : IH_S20_IDENT_MANUFACTURER;
                          // Reference to read Manufacturer identification objects structure
    stGeneralDetails  : IH_S20_IDENT_GENERAL; // Reference to read General identification objects structure
    stSpecialDetails  : IH_S20_IDENT_SPECIAL; // Reference to read Special identification objects structure
    stUseOfTheDevice  : IH_S20_IDENT_USEOFDEVICE;
                       // Reference to read/write Use of the device identification objects
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1.

Step 1: Variable initialization

Program:

```
enBusType    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot  := 1; // Module position
```

Step 2: Program

Program:

```
fbIH_S20ModuleIdentDetails(
```

```
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    Slot:= uiSlot,
    IdentMode:= enIdentMode,
    ManufacturerDetails:= stManufacturerDetails,
    GeneralDetails:= stGeneralDetails,
    SpecialDetails:= stSpecialDetails,
    UseOfTheDevice:= stUseOfTheDevice);
```

Structures to identify (device type plate) an S20 module

- stManufacturerDetails - Manufacturer information (read-only)
- stGeneralDetails - Module - General (read-only)
- stSpecialDetails - Module - Special (read-only)
- stUseOfTheDevice - Module - Special (read/write)

☞ Refer to the chapter “Standard objects” in the data sheet of the S20 module

23.5 Acyclic communication (Acyclic Communication)

23.5.1 IH_S20ComConfiguration

Example program

The example program is used to create a communication handle for S20 "*Type02" function blocks with the **S20BusHandle** input.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig".

TRUE at the "bEnable" input applies the configuration and the function block generates the handle.

☞ The **S20BusHandle** applies to all S20 modules/function blocks at a bus master. A bus master can be an S20 bus coupler or the XMxx control with S20 modules locally connected in series.

Example 1:

Step 1: Variable declaration at the XMxx S20 IO local bus, module at slot 1

Declaration:

Program:

```
PROGRAM ProgIH_S20ComInterfaceType02
VAR
    fbIH_S20ComConfiguration:   IH_S20ComConfiguration;  // Function block instance
    bEnableComConfig:           BOOL; // Function block processing enabled(TRUE - once, edge-controlled)
    bInOperationComConfig:      BOOL; //TRUE: Function Block is working on its actual task
    bErrorComConfig :           BOOL; //TRUE: Indicates an error
    ErrorIDComConfig:           ERROR_CODE; //Class of error
    ErrorIdentComConfig:        ERROR_STRUCT; //Detailed information about error
    enBusType:                  IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:                IH_S20_BUS_CONFIG; //Bus configuration parameters
    dwS20BusHandle:             DWORD; //Bus handle of S20IO configuration

    fbIH_S20ComInterfaceType02: IH_S20ComInterfaceType02; // Function block instance
    bExecute:                   BOOL; // Function block processing enabled(TRUE - once, edge-controlled)
    bDone:                      BOOL; //TRUE: Function Block has completed its task
```

```
    bActive:                    BOOL; //TRUE: Function Block is working on its actual task
    bError:                     BOOL; //TRUE: Indicates an error
    ErrorID:                    ERROR_CODE; //Class of error
    ErrorIdent:                 ERROR_STRUCT; //Detailed information about error
    enSetOpMode:                IH_S20_OP_MODE; //FB Operating mode selection
    uiSlot:                     UINT ;
                                // Module slot address. Bus coupler : 0,
                                // 1st Module after coupler:1 ... 63rd Module after coupler: 63
    stPDIIndex:                 IH_S20_PARAM; //PDI object mapping to  bus parameter
    tTimeout:                   TIME := T#3S; // Timeout delay
    uiNoOfRecBytes:             UINT; // No. of received bytes
    arIndexData:                ARRAY[0..50] OF BYTE;
                                // Address of PDI Index data values.
                                // Common interface for read/write operation
END_VAR
```

Initialization of variables of the function block connected to the XM local and module at slot 1 in the program.

Program:

Program:

```
//Implementation
//==============

// Assignment of BusType to S20IO local
enBusType := IH_S20_BUSTYPE.S20IO;

fbIH_S20ComConfiguration(
    Enable:= bEnableComConfig,
    InOperation=> bInOperationComConfig,
    Error=> bErrorComConfig,
    ErrorID=> ErrorIDComConfig,
    ErrorIdent=> ErrorIdentComConfig,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    S20BusHandle=> dwS20BusHandle);


uiSlot := 1; // Slot no
enSetOpMode := IH_S20_OP_MODE.DEVICETYPE_READ; //Setting Operating mode to read DeviceType

fbIH_S20ComInterfaceType02(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    S20BusHandle:= dwS20BusHandle,
    SetOpMode:= enSetOpMode,
    Slot:= uiSlot,
    PDIIndex:= stPDIIndex,
    IndexDataAdr:= ADR(arIndexData),
    SizeOfIndexData:= SIZEOF(arIndexData),
    Timeout:= tTimeout,
    NoOfRecBytes=> uiNoOfRecBytes);
```

23.5.2 IH_S20ComInterfaceType02

Example program

The example program is used to read the device type of an S20 module connected to an XM control in series.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig".

TRUE at the "bEnable" input applies the configuration and the function block generates the handle.

Example 1:

Step 1: Variable declaration at the XMxx S20 IO local bus, module at slot 1

Declaration:

Program:

```
PROGRAM ProgIH_S20ComInterfaceType02
VAR
    fbIH_S20ComConfiguration:   IH_S20ComConfiguration;  // Function block instance
    bEnableComConfig:           BOOL; // Function block processing enabled( TRUE - once, edge-controlled )
    bInOperationComConfig:      BOOL; //TRUE: Function Block is working on its actual task
    bErrorComConfig :           BOOL; //TRUE: Indicates an error
    ErrorIDComConfig:           ERROR_CODE; //Class of error
    ErrorIdentComConfig:        ERROR_STRUCT; //Detailed information about error
    enBusType:                  IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:                IH_S20_BUS_CONFIG; //Bus configuration parameters
    dwS20BusHandle:             DWORD; //Bus handle of S20IO configuration

    fbIH_S20ComInterfaceType02: IH_S20ComInterfaceType02; // Function block instance
    bExecute:                   BOOL; // Function block processing enabled( TRUE - once, edge-controlled )
    bDone:                      BOOL; //TRUE: Function Block has completed its task
    bActive:                    BOOL; //TRUE: Function Block is working on its actual task
    bError:                     BOOL; //TRUE: Indicates an error
    ErrorID:                    ERROR_CODE; //Class of error
    ErrorIdent:                 ERROR_STRUCT; //Detailed information about error
    enSetOpMode:                IH_S20_OP_MODE; //FB Operating mode selection
    uiSlot:                     UINT ;
                                // Module slot address. Bus coupler : 0,
                                // 1st Module after coupler:1 ... 63rd Module after coupler: 63
    stPDIIndex:                 IH_S20_PARAM; //PDI object mapping to  bus parameter
    tTimeout:                   TIME := T#3S; // Timeout delay
    uiNoOfRecBytes:             UINT; // No. of received bytes
    arIndexData:                ARRAY[0..50] OF BYTE;
                                // Address of PDI Index data values.
                                // Common interface for read/write operation
END_VAR
```

Initialization of variables of the function block connected to the XM local and module at slot 1 in the program.

Program:

Program:

```
// Assignment of BusType to S20IO local
enBusType := IH_S20_BUSTYPE.S20IO;

fbIH_S20ComConfiguration(
    Enable:= bEnableComConfig,
    InOperation=> bInOperationComConfig,
    Error=> bErrorComConfig,
    ErrorID=> ErrorIDComConfig,
    ErrorIdent=> ErrorIdentComConfig,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    S20BusHandle=> dwS20BusHandle);



uiSlot := 1; // Slot no
enSetOpMode := IH_S20_OP_MODE.DEVICETYPE_READ; //Setting Operating mode to read DeviceType

fbIH_S20ComInterfaceType02(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    S20BusHandle:= dwS20BusHandle,
    SetOpMode:= enSetOpMode,
    Slot:= uiSlot,
    PDIIndex:= stPDIIndex,
    IndexDataAdr:= ADR(arIndexData),
    SizeOfIndexData:= SIZEOF(arIndexData),
    Timeout:= tTimeout,
    NoOfRecBytes=> uiNoOfRecBytes);
```

23.5.3 IH_S20ComInterface

Functional description

The function block **IH_S20ComInterface** is the basis for PDI accesses (acyclic communication) with S20 modules via bus master.

Other S20 function blocks internally use the function block **IH_S20ComInterface**.

Structures and enumerations

The enumeration **IH_S20_BUSTYPE** maps the possible field buses.

IH_S20_BUSTYPE		
ENUM	**Value**	**Description**
S20_LOCAL	16#0000	S20 module at the S20 locally at the IndraControl XM
SERCOS	16#0001	S20 module at the bus coupler Sercos
PROFINET	16#0002	S20 module at the bus coupler Profinet
ETHERNETIP	16#0003	S20 module at the bus coupler Ethernet/IP
Profibus	16#0004	S20 module at the bus coupler Profibus
ETHERNET	16#0005	S20 module at the bus coupler Ethernet

The structure **IH_S20_BUS_CONFIG** is used of the field bus-specific addressing of the S20 bus coupler

IH_S20_BUS_CONFIG		
STRUCT	**Data type**	**Description**
Sercos	IH_S20_SERCOS_CONFIG	Sercos bus coupler configuration
Profinet	IH_S20_PROFINET_CONFIG	Profinet bus coupler configuration
Ethernet/IP	IH_S20_ETHERNETIP_CONFIG	Ethernet/IP bus coupler configuration
Profibus	IH_S20_PROFIBUS_CONFIG	Profibus bus coupler configuration
Ethernet	IH_S20_ETHERNET_CONFIG	Ethernet bus coupler configuration

The enumeration **IH_S20_OP_MODE** specifies the action, sends a defined command or reading/sending PDI object.

IH_S20_OP_MODE		
ENUM	**Value**	**Description**
DEVICETYPE_READ	16#00	Reading the module identification **DeviceType**: 0037hex
DIAG_READ	16#01	Reading the diagnostics of the PDI object, diagnostic state (0018hex: **DiagState**)
CONFIG_WRITE	16#02	Writing the parameterization on the PDI object parameter table (0080hex: **ParaTable**)
DIAGRESET_WRITE	16#03	Resetting diagnostics 0019hex **ResetDiag**
OTHER_READ	16#04	Reading any PDI object/index specified at the input "PDIIndex"
OTHER_WRITE	16#05	Writing any PDI object/index specified at the input "PDIIndex"
CONFIG_READ	16#08	Reading the parameterization on the PDI object parameter table (0080hex: **ParaTable**)

The structure **IH_S20_SERCOS_CONFIG** is used to specify the Sercos bus master and the Sercos address.

IH_S20_SERCOS_CONFIG						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
BusMaster	IL_BUSMAS-TER	Bus master	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
SercosAdr	UINT	Sercos address	0 (MLD: local axis) 1 (other sys-tems)	511	0	

The structure **IH_S20_PROFINET_CONFIG** is used to specify the Profinet bus master and the station name.

IH_S20_PROFINET_CONFIG						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
BusMaster	IL_BUSMAS-TER	Bus master	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
IdentStr	STRING	Identification string of the field bus de-vice	----	---	---	

The structure **IH_S20_ ETHERNETIP _CONFIG** is used to specify the Ethernet/IP bus master and the IP address.

IH_S20_ETHERNETIP_CONFIG						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
BusMaster	IL_BUSMAS-TER	Bus master	IL_BUSMAS-TER_0	IL_BUSMAS-TER_4	IL_BUSMAS-TER_0	Rising edge at "Execute"
CIPPath	STRING	Address of the CIP device "aaa.bbb.ccc.ddd"	--	--	--	

The structure **IH_S20_PROFIBUS_CONFIG** is used to specify the Profibus bus master, the segment and the address.

IH_S20_PROFIBUS_CONFIG						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
Master	BYTE	DP master identifi-cation	---	---	---	Rising edge at "Execute"
Segment	BYTE	Number of the DP segment (0)	---	---	---	
Station	BYTE	Number of the DP slave (bus address)	---	---	---	

The structure **IH_S20_ETHERNET_CONFIG** is used to specify the IP address and the port.

IH_S20_ETHERNET_CONFIG						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
Address	STRING	Linked IP address	--	--	‘’	Rising edge at "Execute"
Port	UINT	TCP port of the Modbus TCP server	0	65535	502	

The structure **IH_S20_PARAM** is used to map the PDI index on field bus-specific parameters.

IH_S20_PARAM		
STRUCT	**Data type**	**Description**
Sercos	IH_S20_SIII_PARAM	Sercos parameter mapping
Profinet	IH_S20_PN_PARAM	Profinet index mapping
Ethernet/IP	IH_S20_ENIP_PARAM	Ethernet/IP attribute mapping
Profibus	IH_S20_PB_PARAM	Profibus index mapping
Ethernet	IH_S20_ETH_PARAM	ModbusTCP register mapping

The structure **IH_S20_SIII_PARAM** is used to map the PDI index on Sercos parameters.

IH_S20_SIII_PARAM						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
Para-mType	IL_SIII_PAR-AM_TYPE	Standard (S) or product-specific (P) IDN	IL_S_PARAM	IL_P_PARAM	IL_S_PARAM	Rising edge at "Execute"
Set	USINT	Parameter set	16#00	16#FF	16#00	
Nbr	UINT	Number of the data function blocks (index)	16#00	16#FFFF	16#00	
StructE-lem	USINT	Structural element (subindex)	16#00	16#FF	16#00	

The structure **IH_S20_PN_PARAM** is used to map the PDI index on Profinet parameters.

IH_S20_PN_PARAM						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
Index	UINT	Index of the address data set	16#00	16#FFFF	16#00	Rising edge at "Execute"
Subslot	UINT	Addressing the sub-module	16#01	16#8FFF	16#00	

The structure **IH_S20_ENIP_PARAM** is used to map the PDI index on Ethernet/IP attributes.

IH_S20_ENIP_PARAM						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
Class	UINT	ClassCode -16#69 for PDI access	16#01	16#FF	16#69	Rising edge at "Execute"
Instance	UINT	Instance number (slot no.)	16#00	16#FF	16#00	
Attributes	UINT	Attribute ID (index)	16#01	16#FF	16#01	

The structure **IH_S20_PB_PARAM** is used to map the PDI index on Profibus parameters.

IH_S20_PB_PARAM						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
Index	INT	Process data index	16#00	16#7FFF	16#00	Rising edge at "Execute"

The structure **IH_S20_ETH_PARAM** is used to map the PDI index on Modbus TCP registers.

IH_S20_ETH_PARAM						
STRUCT	**Data type**	**Description**	**Min. value**	**Max. value**	**Default value**	**Application**
Index	UINT	Data index register	16#00	16#FFFF	16#00	Rising edge at "Execute"

Example program

The example program describes the reading/writing of PDI objects using different field buses.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "Execute" input applies the configuration and the function block executes the selected PDI action in "enSetOpMode".

Declaration:

Program:

```
PROGRAM ProgIH_S20ComInterface
VAR
    fbIH_S20ComInterface: IH_S20ComInterface; // Function block instance
    bExecute: BOOL; // Function block processing enabled( TRUE - once, edge-controlled )
    bDone: BOOL; //TRUE: Function Block has completed its task
    bActive: BOOL; //TRUE: Function Block is working on its actual task
    bError: BOOL; //TRUE: Indicates an error
    ErrorID: ERROR_CODE; //Class of error
    ErrorIdent: ERROR_STRUCT; //Detailed information about error
    enBusType: IH_S20_BUSTYPE  //Bus type selection
    stBusConfig: IH_S20_BUS_CONFIG; //Bus configuration parameters
    enSetOpMode: IH_S20_OP_MODE; //FB Operating mode selection
    uiSlot: UINT ; //Module slot address.Bus coupler:0,
                //1st Module after coupler:1...63rd Module after coupler:63
    stPDIIndex: IH_S20_PARAM; //PDI object mapping to  bus parameter
    tTimeout: TIME := T#3S; // Timeout delay
    uiNoOfRecBytes: UINT; // No. of received bytes
    arIndexData: ARRAY[0..50] OF BYTE; //Address of PDI Index data values.
                                //Common interface for read/write operation
END_VAR
```

Example 1:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 to read the DeviceType.

Variable initialization:

Program:

```
enBusType:= IH_S20_BUSTYPE.SERCOS;
enSetOpMode:= IH_S20_OP_MODE.DEVICETYPE_READ;
uiSlot := 1;
```

Example 2:

Function block initialization to read the diagnostics at the Profinet bus coupler.

Variable initialization:

Program:

```
enBusType:= IH_S20_BUSTYPE.PROFINET;
enSetOpMode:= IH_S20_OP_MODE.DIAG_READ;
uiSlot := 1;
```

Example 3:

Function block initialization to write the configuration for the S20-AI-4-I at the Profibus bus coupler.

Variable declaration:

```
arIndexData: ARRAY [0..14] OF BYTE := [0, 4(4), 9(0)];
```

Variable initialization:

Program:

```
enBusType:=  IH_S20_BUSTYPE.PROFIBUS;
enSetOpMode:= IH_S20_OP_MODE.CONFIG_WRITE;
stBusConfig.Profibus.Master := 0;
stBusConfig.Profibus.Segment := 0;
stBusConfig.Profibus.Station := 2;
uiSlot := 1;
```

Example 4:

Function block initialization to reset the diagnostics of an S20 module at the Ethernet/IP bus coupler.

Variable initialization:

Program:

```
enBusType:= IH_S20_BUSTYPE.ETHERNETIP;
enSetOpMode:= IH_S20_OP_MODE. DIAGRESET_WRITE;
uiSlot := 1;
```

Example 5:

Function block initialization to read the "Location Name" (index: 0014hex) at the Sercos bus coupler.

Variable declaration:

```
arIndexData: ARRAY [0..58] OF BYTE;
```

Variable initialization:

Program:

```
enBusType:= IH_S20_BUSTYPE.SERCOS;
enSetOpMode:= IH_S20_OP_MODE.OTHER_READ;
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69;

stPDIIndex.Sercos.ParamType := IL_SIII_PARAM_TYPE.IL_P_PARAM;
stPDIIndex.Sercos.Set := 1;
stPDIIndex.Sercos.Nbr := 16#14;
stPDIIndex.Sercos.StructElem := 0;
uiSlot := 1;
```

Example 6:

Function block initialization to write the "Location Name" (index: 0014hex) at the Profinet bus coupler.

Variable declaration:

```
arIndexData: ARRAY [0..58] OF BYTE := [82, 57,49, 55(0)];
```

Variable initialization:

Program:

```
enBusType:= IH_S20_BUSTYPE.PROFINET;
enSetOpMode:= IH_S20_OP_MODE.OTHER_WRITE;
stBusConfig.Profinet.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Profinet.IdentStr := 's20-pn-bk'

stPDIIndex.Profinet.Index := 16#14;
stPDIIndex.Profinet.SubSlot := 1;
uiSlot := 1;
```

Program:

Program:

```
fbIH_S20ComInterface(
    Execute:= bExecute,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    SetOpMode:= enSetOpMode,
    Slot:= uiSlot,
    PDIIndex:= stPDIIndex,
    IndexDataAdr:= ADR(arIndexData),
    SizeOfIndexData:= SIZEOF(arIndexData),
    Timeout:= tTimeout,
    NoOfRecBytes=> uiNoOfRecBytes);
```

23.6 Analog input modules (AnalogIO)

23.6.1 IH_S20AI4IU

Example program

The example program is used to read the analog values of the modules **S20-AI-4-U** and **S20-AI-4-I**.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Example 1:

Step 1: Module configuration in IndraWorks Engineering for channel 1 to measure the analog voltage of 10V, filter 30Hz and 16 samples

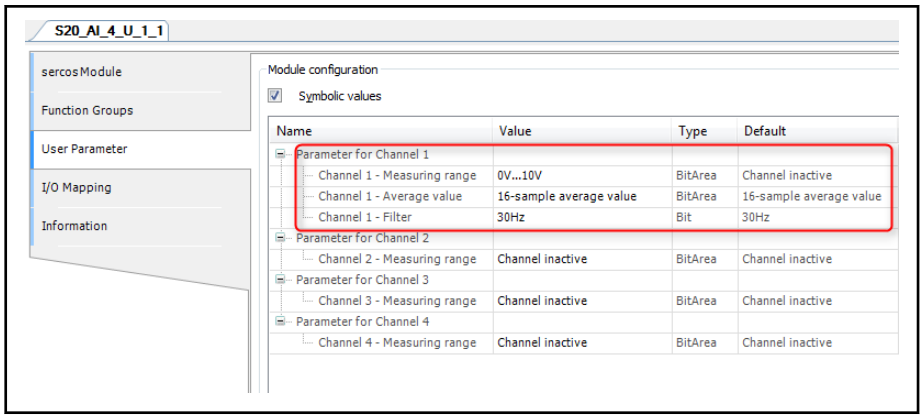


Fig. 23-2: Example 1, step 1

Step 2: Variable declaration for the module S20-AI-4-U at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AI4IU
VAR
    fbIH_S20AI4IU:   IH_S20AI4IU;
    bEnable:         BOOL; //TRUE: Function is enabled
    bInOperation:    BOOL; //TRUE: Function Block is working on its actual task
    bError:          BOOL; //TRUE: Indicates an error
    ErrorID:         ERROR_CODE; //Class of error
    ErrorIdent:      ERROR_STRUCT; //Detailed information about error
    enBusType:       IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:     IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:          UINT; //Module slot address.Bus coupler:0,
                  //1st Module after coupler:1...63rd Module after coupler:63
    arChannelConfig: ARRAY[1..4] OF  IH_S20_AI_CONFIG; //Structure for channel configuration
    arInProcData:    ARRAY[1..4] OF WORD; //Input Process variables
    arOutputData:    ARRAY[1..4] OF IH_S20_AI_DATA; //Structure for Output Data
    rCh1_Voltage:    REAL; // Ch-1 Voltage input value
END_VAR
```

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
//Ch-1 Configuration
arChannelConfig[1].HighLimit := 10.0; //10V
arChannelConfig[1].LowLimit  := 0.0; //0v
```

Program:

Program:

```
fbIH_S20AI4IU (
    Enable        := bEnable,
    InOperation=> bInOperation,
    Error=>       bError,
    ErrorID=>     ErrorID,
    ErrorIdent=>  ErrorIdent,
    BusType       := enBusType,
    BusConfig     := stBusConfig,
```

```
    Slot          := uiSlot,
    ChannelConfig := arChannelConfig,
    InProcData    := arInProcData,
    OutputData    := arOutputData);
```

Reading analog input values:

```
rCh1_Voltage:= arOutputData[1].NormalizedValue;
```

Example 2:

Step 1: Module configuration in IndraWorks Engineering for channel 1 to measure the current between 0 mA and 20 mA, filter 30 Hz and 16 samples

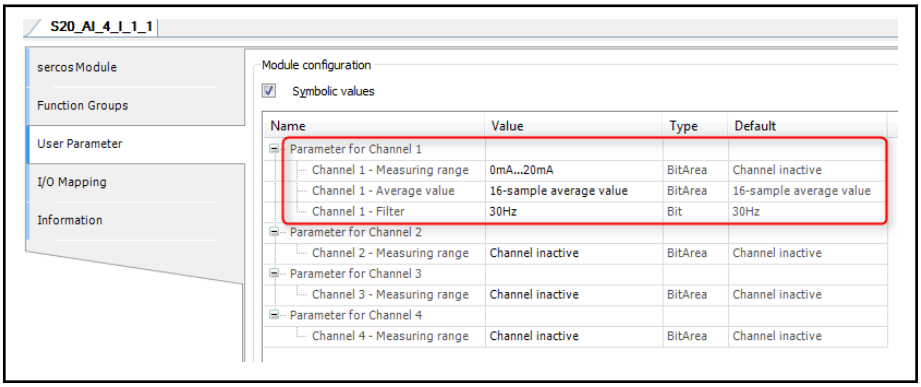


Fig. 23-3: Example 2, step 1

Step 2: Variable declaration for the module S20-AI-4-I at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AI4IU
VAR
    fbIH_S20AI4IU:   IH_S20AI4IU;
    bEnable:         BOOL; //TRUE: Function is enabled
    bInOperation:    BOOL; //TRUE: Function Block is working on its actual task
    bError:          BOOL; //TRUE: Indicates an error
    ErrorID:         ERROR_CODE; //Class of error
    ErrorIdent:      ERROR_STRUCT; //Detailed information about error
    enBusType:       IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:     IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:          UINT; //Module slot address. Bus coupler : 0,
                        //1st Module after coupler:1 ... 63rd Module after coupler: 63
    arChannelConfig: ARRAY[1..4] OF  IH_S20_AI_CONFIG; //Structure for channel configuration
    arInProcData:    ARRAY[1..4] OF WORD; //Input Process variables
    arOutputData:    ARRAY[1..4] OF IH_S20_AI_DATA; //Structure for Output Data
    rCh1_Voltage:    REAL; // Ch-1 Voltage input value
END_VAR
```

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
//Ch-1 Configuration
arChannelConfig[1].HighLimit := 10.0; //10V
arChannelConfig[1].LowLimit  := 0.0; //0v
```

Program:

Program:

```
fbIH_S20AI4IU (
    Enable        := bEnable,
    InOperation=> bInOperation,
    Error=>       bError,
    ErrorID=>     ErrorID,
    ErrorIdent=>  ErrorIdent,
    BusType       := enBusType,
    BusConfig     := stBusConfig,
    Slot          := uiSlot,
    ChannelConfig := arChannelConfig,
    InProcData    := arInProcData,
    OutputData    := arOutputData);
```

Program:

```
fbIH_S20AI4IU (
    Enable        := bEnable,
    InOperation=> bInOperation,
    Error=>       bError,
    ErrorID=>     ErrorID,
    ErrorIdent=>  ErrorIdent,
    BusType       := enBusType,
    BusConfig     := stBusConfig,
    Slot          := uiSlot,
    ChannelConfig := arChannelConfig,
    InProcData    := arInProcData,
    OutputData    := arOutputData);
```

Reading analog input values:

```
rCh1_Current:= arOutputData[1].NormalisedValue;
```

Step3: Step 3 is valid for the examples 1 and 2

Linking process data in the I/O mapping at the module in IndraWorks:

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-AI-4-I. It is identical for the module S20-AI-4-U.

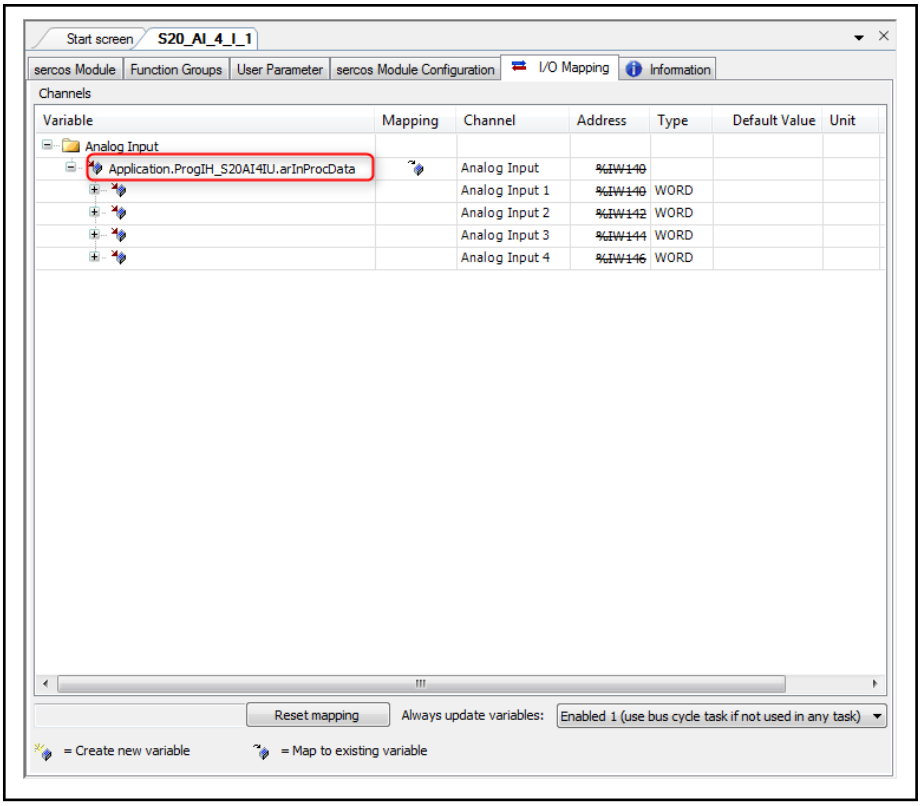


Fig. 23-4: Example 2, step 3

23.6.2 IH_S20AI8

Example program

The example program is used to read the analog values of the S20-AI-8 module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for channel 1 to measure the analog voltage of +-5V, filter 30Hz and 16 samples

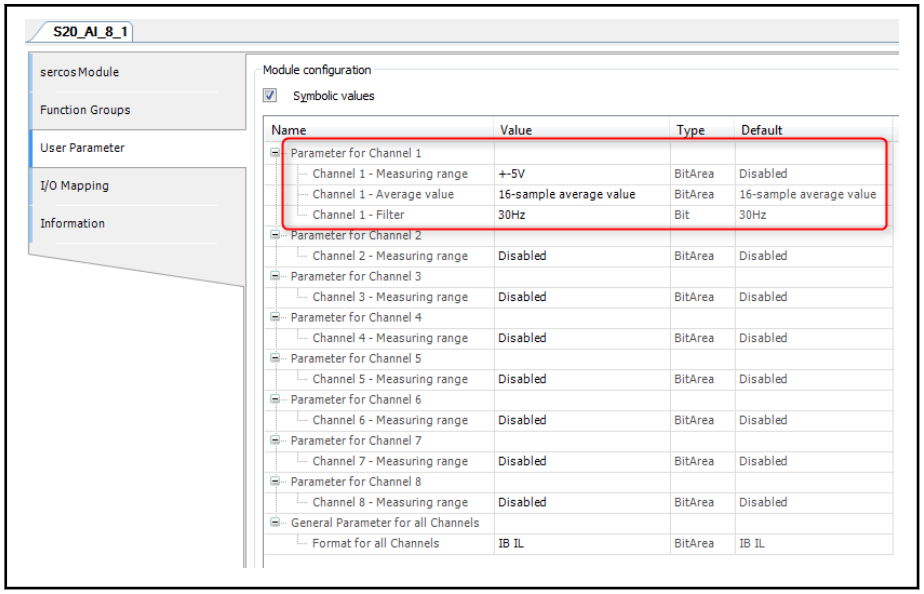


Fig. 23-5: Step 1

Step 2: Variable declaration for the module S20-AI-8 at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AI8
VAR
    fbIH_S20AI8:        IH_S20AI8; // FB Instance
    bEnable:            BOOL; //TRUE: Function is enabled
    bInOperation:       BOOL; //TRUE: Function Block is working on its actual task
    bError:             BOOL; //TRUE: Indicates an error
    ErrorID:            ERROR_CODE; //Class of error
    ErrorIdent:         ERROR_STRUCT; //Detailed information about error
    enBusType:          IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:        IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:             UINT; //Module slot address. Bus coupler : 0,
                           //1st Module after coupler:1 ... 63rd Module after coupler: 63
    arChannelConfig:    ARRAY[1..8] OF  IH_S20_AI_CONFIG; //Structure for channel configuration
    arInProcData:       ARRAY[1..8] OF WORD; //Input Process variables
    arOutputData:       ARRAY[1..8] OF IH_S20_AI_DATA; //Structure for Output Data
    rCh1_Voltage:       REAL; // Ch-1 voltage value
END_VAR
```

Variable declaration for the module S20-AI-8 at the Sercos bus coupler:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position

arChannelConfig[1].HighLimit := 5.0; //+5V
arChannelConfig[1].LowLimit  := -5.0; //-5V
```

Program:

Program:

```
fbIH_S20AI8 (
    Enable        := bEnable,
    InOperation=> bInOperation,
    Error=>       bError,
    ErrorID=>     ErrorID,
    ErrorIdent=>  ErrorIdent,
    BusType       := enBusType,
    BusConfig     := stBusConfig,
    Slot          := uiSlot,
    ChannelConfig := arChannelConfig,
    InProcData    := arInProcData,
    OutputData    := arOutputData);
```

Reading analog input values:

```
rCh1_Voltage := arOutputData[1].NormalisedValue;
```

Step 3: Linking process data in the I/O mapping at the module in IndraWorks:

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-AI-8.

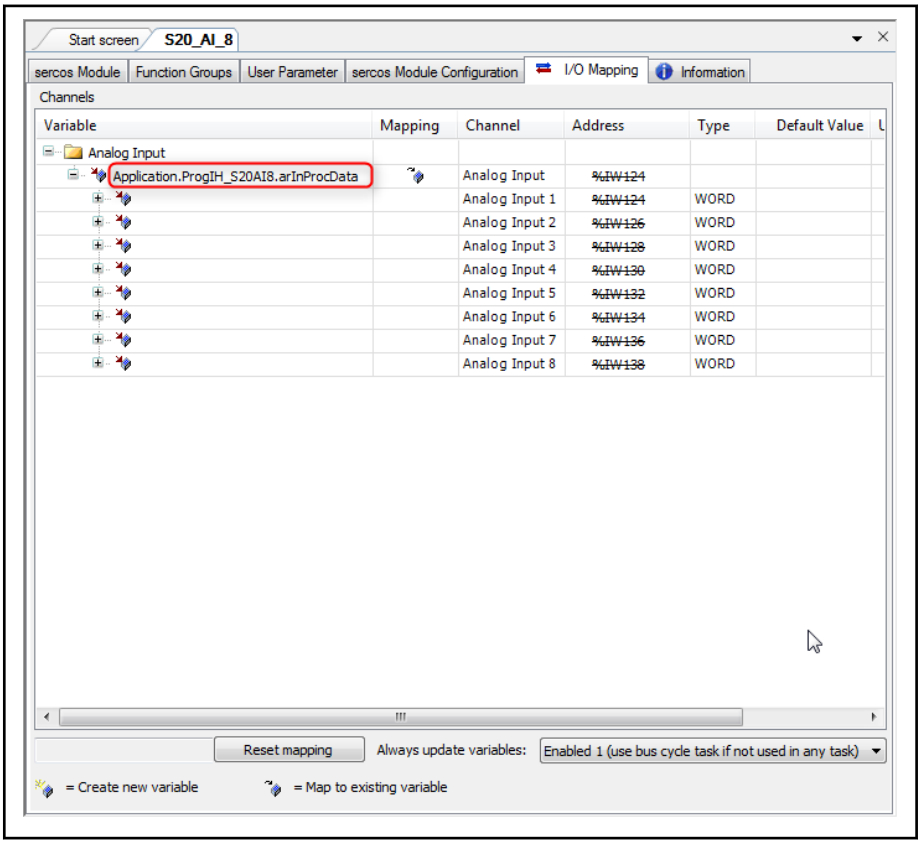


Fig. 23-6: Step 3

23.6.3 IH_S20AI4RTD

Example program

The example program is used to read the analog values of the S20-AI-4-RTD module.

The field bus used is defined at the input "BusType" using the variable "en-BusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for channel 1 to measure a PT-100 sensor with the resolution 0.1 °C in three-wire technology and a resistor of 100 Ohms, filter 120 ms

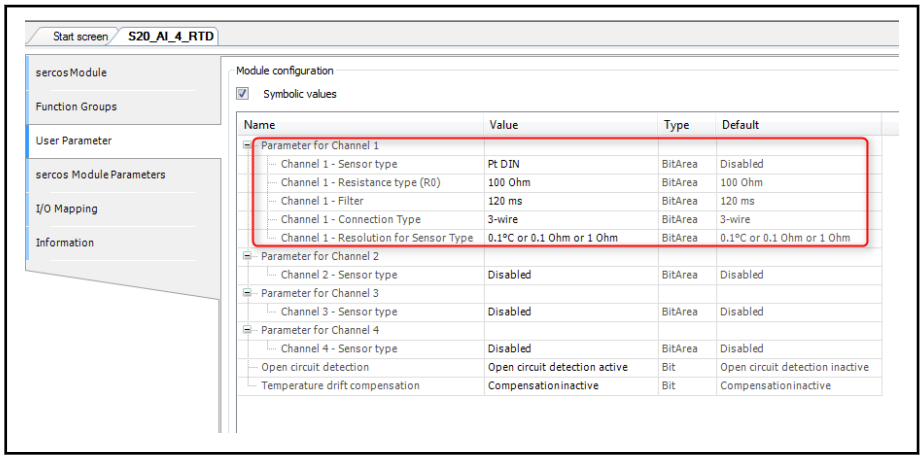


Fig. 23-7: Step 1

Step 2: Variable declaration for the module S20-AI-4-RTD at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AI4RTD
VAR
    fbIH_S20AI4RTD:    IH_S20AI4RTD; // FB Instance
    bEnable:           BOOL; //TRUE: Function is enabled
    bInOperation:      BOOL; //TRUE: Function Block is working on its actual task
    bError:            BOOL; //TRUE: Indicates an error
    ErrorID:           ERROR_CODE; //Class of error
    ErrorIdent:        ERROR_STRUCT; //Detailed information about error
    enBusType:         IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:       IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:            UINT; //Module slot address. Bus coupler : 0,
                          //1st Module after coupler:1 ... 63rd Module after coupler: 63
    arInProcData:      ARRAY[1..4] OF WORD; //Input Process variables
    arOutputData:      ARRAY[1..4] OF IH_S20_TEMP_DATA; //Structure for Output Data
    rTemperature:      REAL; // read Temperature value
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
```

Program:

Program:

```
fbIH_S20AI4RTD (
```

```
Enable          := bEnable,
InOperation=>   bInOperation,
Error=>         bError,
ErrorID=>       ErrorID,
ErrorIdent=>    ErrorIdent,
BusType         := enBusType,
BusConfig       := stBusConfig,
Slot            := uiSlot,
InProcData      := arInProcData,
OutputData      := arOutputData);
```

Reading analog input values:

```
rTemperature := arOutputData[1].NormalizedValue;
```

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-AI-8-RTD.

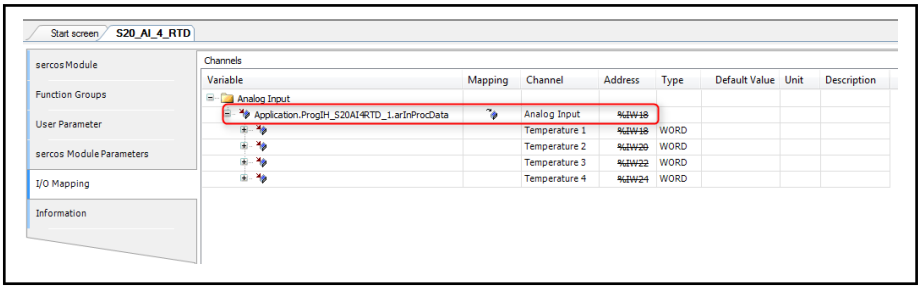


Fig. 23-8: Step 3

23.6.4 IH_S20AI4UTH

Example program

The example program is used to read the analog values of the S20-AI-4-UTH module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for channel 1 to measure a K-type with the resolution 0.1 °C and the filter 120 ms

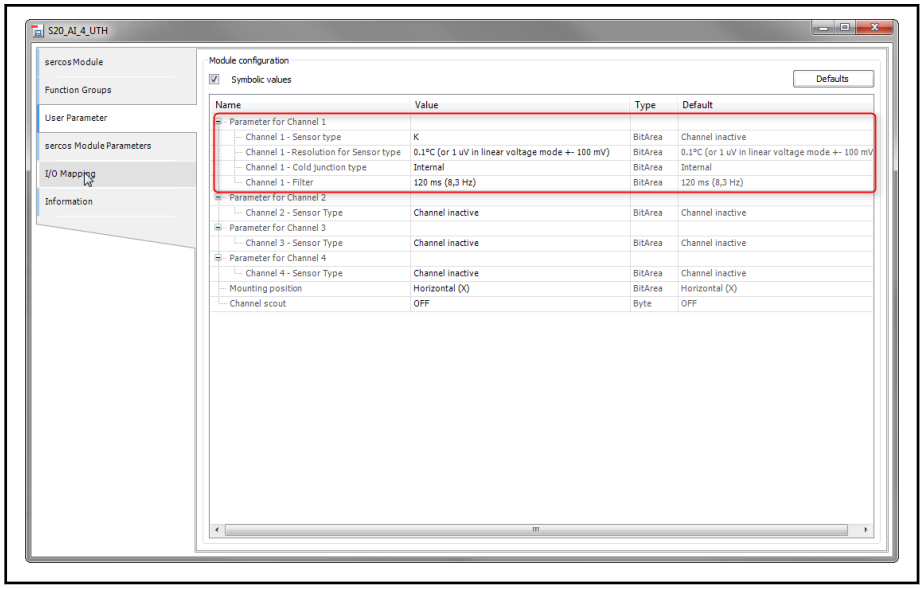


Fig. 23-9: Step 1

Step 2: Variable declaration for the module S20-AI-8-UTH at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AI4UTH
VAR
    fbIH_S20AI4UTH:   IH_S20AI4UTH; // FB Instance
    bEnable:          BOOL; //TRUE: Function is enabled
    bInOperation:     BOOL; //TRUE: Function Block is working on its actual task
    bError:           BOOL; //TRUE: Indicates an error
    enErrorID:        ERROR_CODE; //Class of error
    stErrorIdent:     ERROR_STRUCT; //Detailed information about error
    stBusType:        IH_S20_BUSTYPE; ; //Bus type selection
    stBusConfig:      IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:           UINT := 1; //Module slot address. Bus coupler : 0,
                        //1st Module after coupler:1 ... 63rd Module after coupler: 63
    arInProcData:     ARRAY [1..5] OF WORD; //Input process data
    arOutputData:     ARRAY [1..4] OF IH_S20_TEMP_DATA; //Contains normalized value
                                                //and diagnostic code for channels 1…4.
    arOutProdata_0:   WORD; //Output process data(word 0)
    stAnalogData:     IH_S20_TEMP_DATA; //Contains normalized value
                                //and diagnostic code of analog voltage input ±5V
    rTemperature:     REAL; // Read temperature value
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
```

Program:

Program:

```
fbIH_S20AI4UTH (
    Enable            := bEnable,
    InOperation=>     bInOperation,
    Error=>           bError,
    ErrorID=>         enErrorID,
    ErrorIdent=>      stErrorIdent,
    BusType           := stBusType,
    BusConfig         := stBusConfig,
    Slot              := uiSlot,
    InProcData        := arInProcData,
    OutputData        := arOutputData,
    OutProcData_0=>   arOutProdata_0,
    AnalogData=>      stAnalogData);
```

Reading input values:

```
rTemperature := arOutputData[1].NormalisedValue;
```

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module 20-AI-4-UTH.

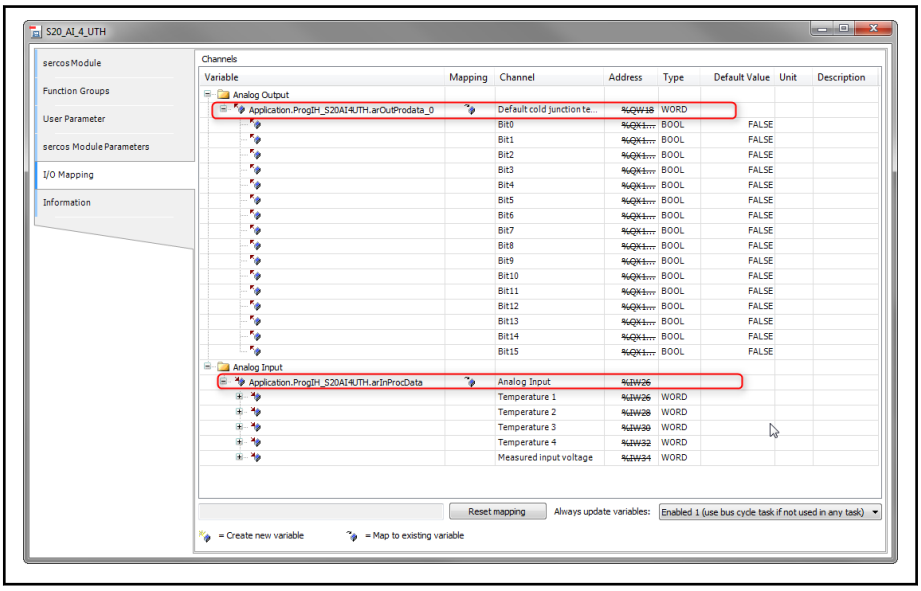


Fig. 23-10: Step 3

23.6.5 IH_S20AI8RTD

Example program

The example program is used to read the analog values of the S20-AI-8-RTD module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for channel 1 to measure a PT-100 sensor with the resolution 0.1 °C in three-wire technology and a resistor of 100 Ohms, filter 120 ms

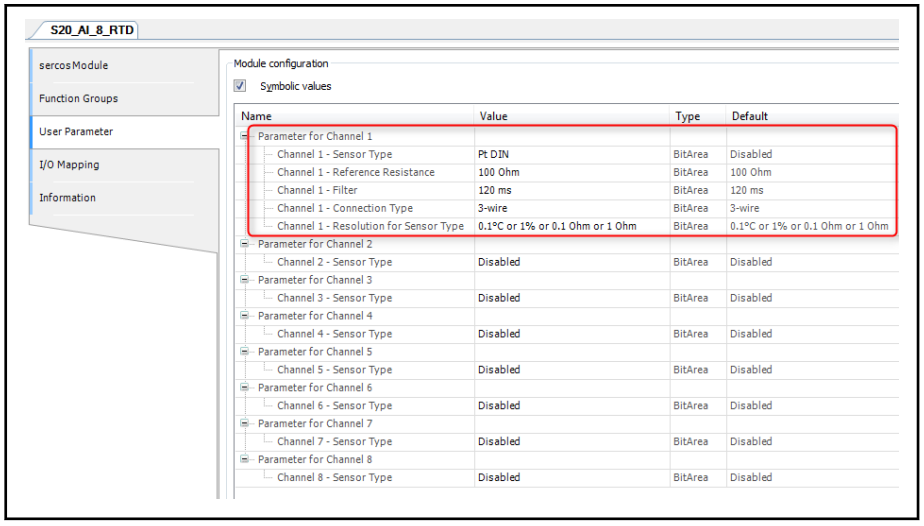


Fig. 23-11: Step 1

Step 2: Variable declaration for the module S20-AI-8-RTD at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AI8RTD
VAR
    fbIH_S20AI8RTD:  IH_S20AI8RTD; // FB Instance
    bEnable:         BOOL; //TRUE: Function is enabled
    bInOperation:    BOOL; //TRUE: Function Block is working on its actual task
    bError:          BOOL; //TRUE: Indicates an error
    ErrorID:         ERROR_CODE; //Class of error
    ErrorIdent:      ERROR_STRUCT; //Detailed information about error
    enBusType:       IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:     IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:          UINT; //Module slot address. Bus coupler : 0,
                        //1st Module after coupler:1 ... 63rd Module after coupler: 63
    arInProcData:    ARRAY[1..8] OF WORD; //Input Process variables
    arOutputData:    ARRAY[1..8] OF IH_S20_TEMP_DATA; //Structure for Output Data
    rTemperature:    REAL; // read Temperature value
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
```

Program:

Program:

```
fbIH_S20AI8RTD (
    Enable          := bEnable,
    InOperation=>   bInOperation,
    Error=>         bError,
    ErrorID=>       ErrorID,
    ErrorIdent=>    ErrorIdent,
```

```
    BusType         := enBusType,
    BusConfig       := stBusConfig,
    Slot            := uiSlot,
    InProcData      := arInProcData,
    OutputData      := arOutputData);
```

Reading analog input values:

```
rTemperature := arOutputData[1].NormalisedValue;
```

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-AI-8-RTD.

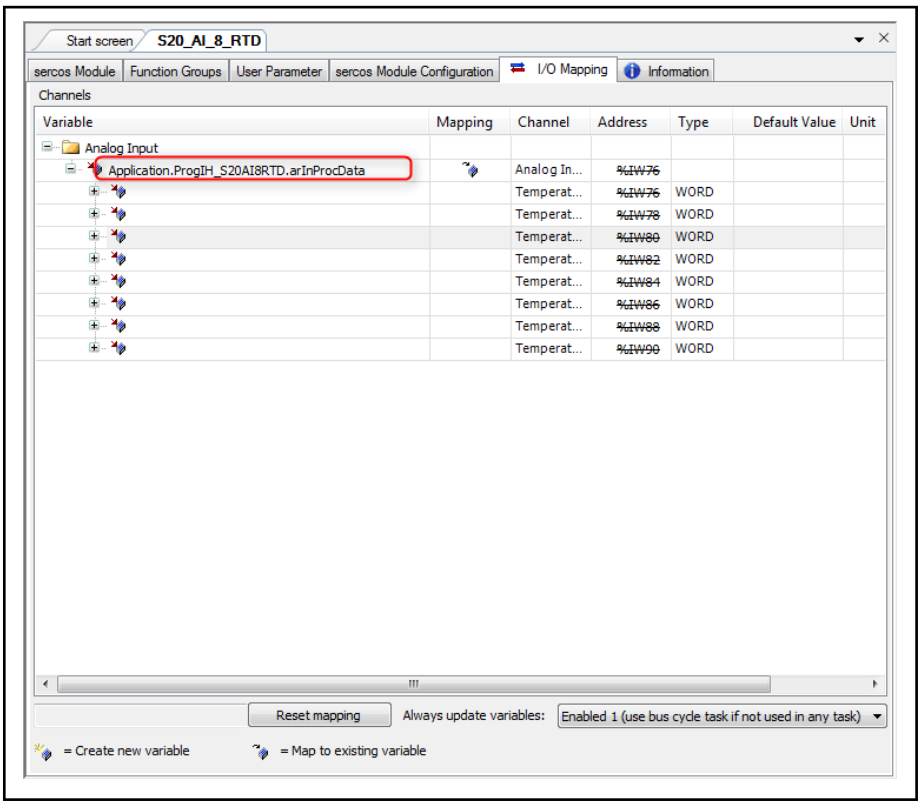


Fig. 23-12: Step 3

23.6.6 IH_S20AI8UTH

Example program

The example program is used to read the analog values of the S20-AI-8-UTH module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for channel 1 to measure a K-type with the resolution 0.1 °C and the filter 120 ms

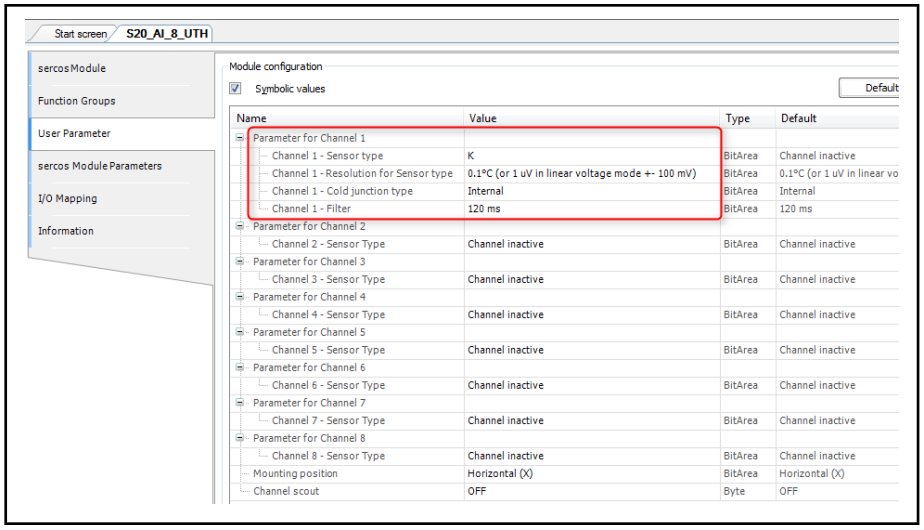


Fig. 23-13: Step 1

Step 2: Variable declaration for the module S20-AI-8-UTH at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AI8UTH
VAR
    fbIH_S20AI8UTH:   IH_S20AI8UTH; // FB Instance
    bEnable:          BOOL; //TRUE: Function is enabled
    bInOperation:     BOOL; //TRUE: Function Block is working on its actual task
    bError:           BOOL; //TRUE: Indicates an error
    enErrorID:        ERROR_CODE; //Class of error
    stErrorIdent:     ERROR_STRUCT; //Detailed information about error
    stBusType:        IH_S20_BUSTYPE; ; //Bus type selection
    stBusConfig:      IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:           UINT := 1; //Module slot address. Bus coupler : 0,
                          //1st Module after coupler:1 ... 63rd Module after coupler: 63

    arInProcData:     ARRAY [1..9] OF WORD; //Input process data
    arOutputData:     ARRAY [1..8] OF IH_S20_TEMP_DATA; //Contains the normalized value
                                                  //and the diagnostic code for channels 1…8.
    arOutProdata_0:   WORD; //Output process data(word 0)
    stAnalogData:     IH_S20_TEMP_DATA; //Contains the normalized value
                                 //and diagnostic code of analog voltage input ±5V
    rTemperature:     REAL; // Read temperature value
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
```

Program:

Program:

```
fbIH_S20AI8UTH (
    Enable      := bEnable,
```

```
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> enErrorID,
    ErrorIdent=> stErrorIdent,
    BusType     := stBusType,
    BusConfig   := stBusConfig,
    Slot        := uiSlot,
    InProcData  := arInProcData,
    OutputData  := arOutputData,
    OutProcData_0=> arOutProdata_0,
    AnalogData=> stAnalogData);
```

Reading input values:

```
rTemperature := arOutputData[1].NormalisedValue;
```

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-AI-8-UTH.

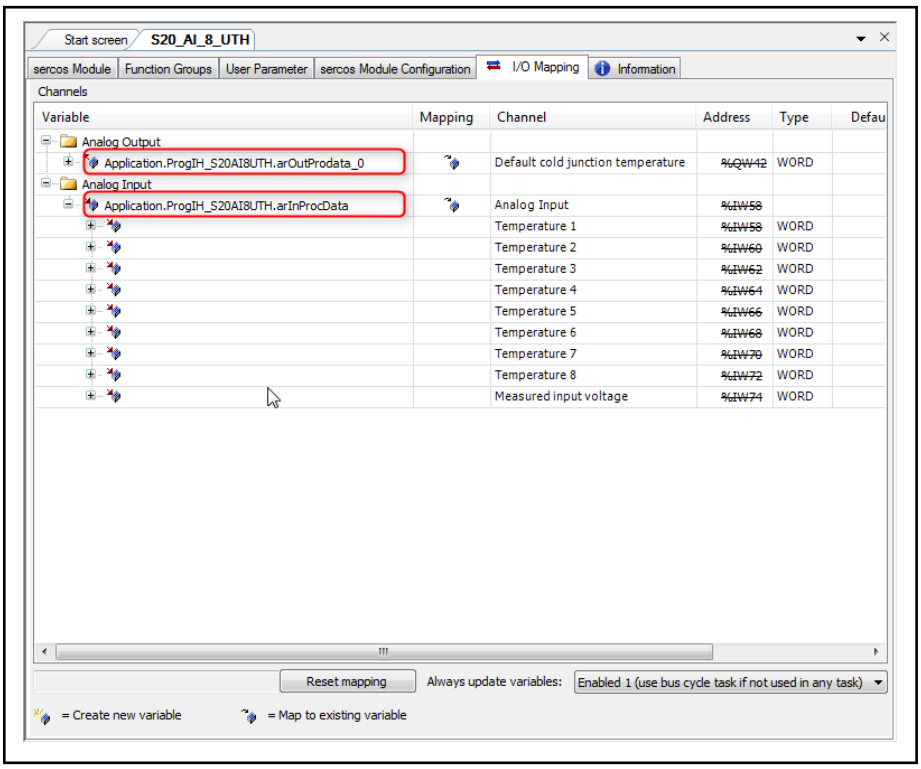


Fig. 23-14: Step 3

23.7 Analog output modules (AnalogIO)

23.7.1 IH_S20AO8

Example program

The example program is used to write the analog values of the S20-AO-8 module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for channel 1 to output 0-10 V and channel -2 of +/-5 V

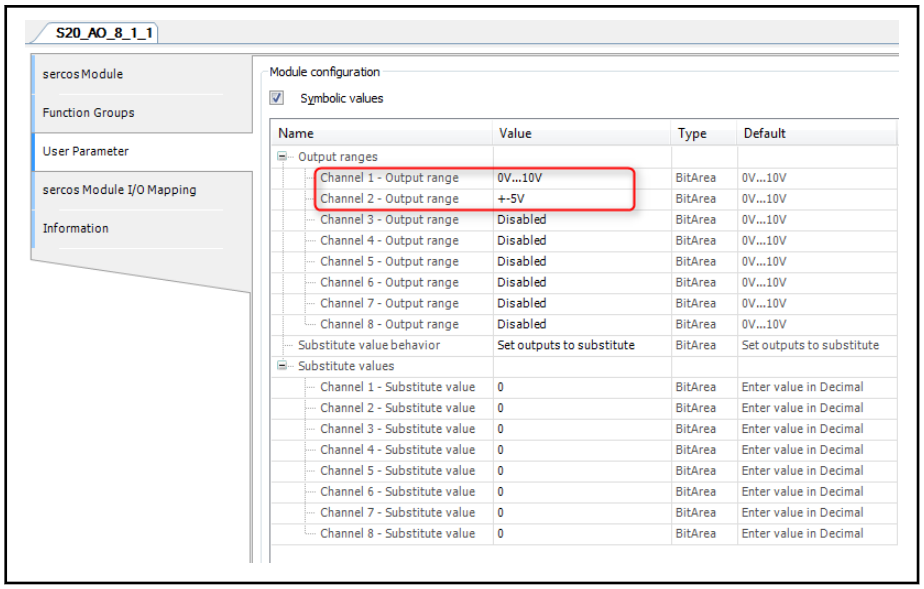


Fig. 23-15: Step 1

Step 2: Variable declaration for the module S20-AO-8 at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AO8
VAR
    fbIH_S20AO8:       IH_S20AO8; // FB Instance
    bEnable:           BOOL; //TRUE: Function is enabled
    bInOperation:      BOOL; //TRUE: Function Block is working on its actual task
    bError:            BOOL; //TRUE: Indicates an error
    ErrorID:           ERROR_CODE; //Class of error
    ErrorIdent:        ERROR_STRUCT; //Detailed information about error
    enBusType:         IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:       IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:            UINT; //Module slot address. Bus coupler : 0,
                          //1st Module after coupler:1 ... 63rd Module after coupler: 63
    arChannelConfig:   ARRAY[1..8] OF  IH_S20_AO_CONFIG; //Structure for channel configuration
    bBusy:             BOOL; //Busy Signal used for PDI handling in sequence
    arOutProcData:     ARRAY [1..8] OF WORD; // Outprocess data
    DiagCode:          ARRAY [1..8] OF STRING(14); //Channel diagnostic information
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program with specification of the limits per channel.

Step 3: Variable initialization

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
//Ch-1 Configuration
arChannelConfig[1].HighLimit := 10.0;
arChannelConfig[1].LowLimit  :=  0.0;
//Ch-2 Configuration
```

```
arChannelConfig[2].HighLimit := 5.0;
arChannelConfig[2].LowLimit  :=  -5.0;
```

Step 4: Program:

Program:

```
fbIH_S20AO8(
    Enable          := bEnable,
    InOperation=>   bInOperation,
    Error=>         bError,
    ErrorID=>       ErrorID,
    ErrorIdent=>    ErrorIdent,
    BusType         := enBusType,
    BusConfig       := stBusConfig,
    Slot            := uiSlot,
    ChannelConfig   := arChannelConfig,
    OutProcData     := arOutProcData,
    DiagCode        := DiagCode);
```

Outputting the analog output value:

```
arChannelConfig[1].NormalizedValue := 10.0;
arChannelConfig[2].NormalizedValue := -5.0;
```

Step 5: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-AO-8.

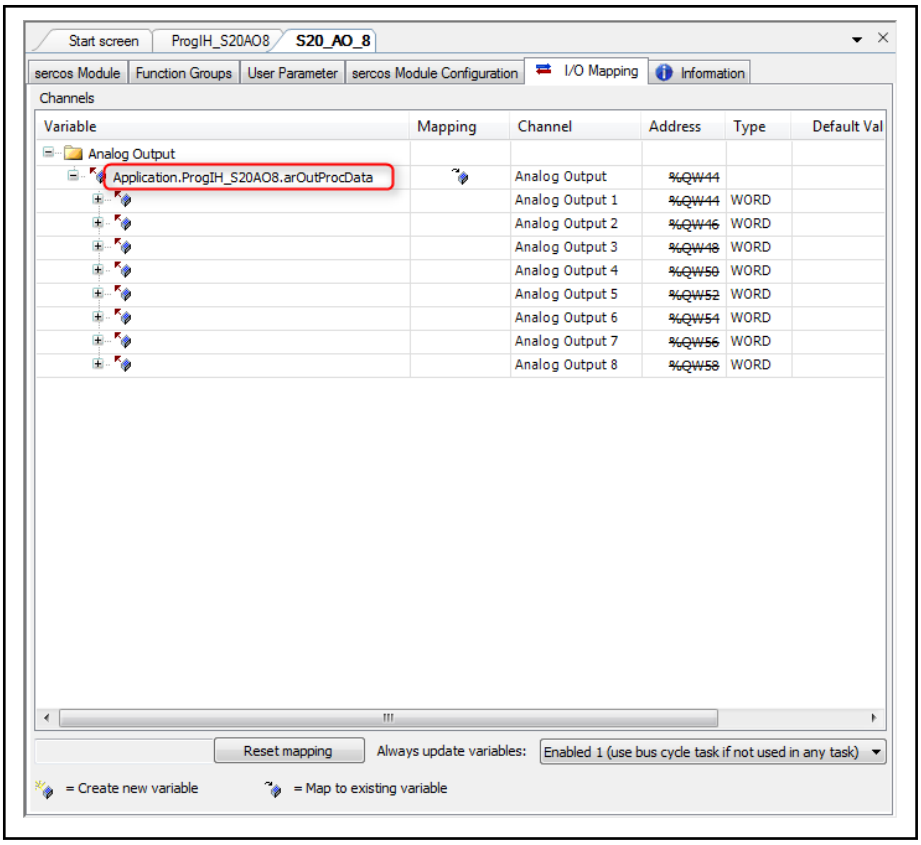


Fig. 23-16: Step 5

23.7.2 IH_S20AO4

Example program

The example program is used to write the analog values of the S20-AO-4 module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for channel 1 to output 0-10 V and channel -2 of +/-5 V

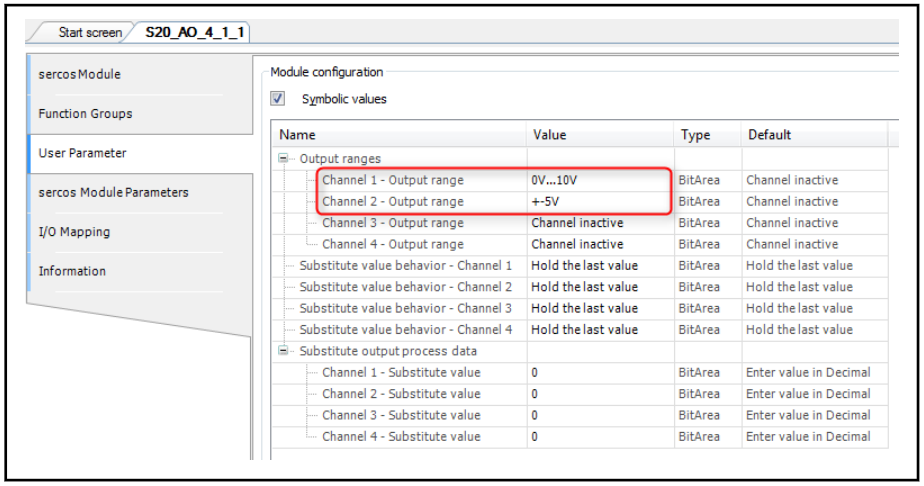


Fig. 23-17: Step 1

Step 2: Variable declaration for the module S20-AO-4 at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20AO4
VAR
    fbIH_S20AO4:      IH_S20AO4; // FB Instance
    bEnable:          BOOL; //TRUE: Function is enabled
    bInOperation:     BOOL; //TRUE: Function Block is working on its actual task
    bError:           BOOL; //TRUE: Indicates an error
    ErrorID:          ERROR_CODE; //Class of error
    ErrorIdent:       ERROR_STRUCT; //Detailed information about error
    enBusType:        IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:      IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:           UINT; //Module slot address. Bus coupler : 0,
                        //1st Module after coupler:1 ... 63rd Module after coupler: 63
    arChannelConfig: ARRAY[1..4] OF  IH_S20_AO_CONFIG; //Structure for channel configuration
    bBusy:            BOOL; //Busy Signal used for PDI handling in sequence
    arOutProcData:    ARRAY [1..4] OF WORD; //Outprocess data
    DiagCode:         ARRAY [1..4] OF STRING(14); //Diagnostics message
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the analog module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program with specification of the limits per channel.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
```

```
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position

//Ch-1 Configuration
arChannelConfig[1].HighLimit := 10.0;
arChannelConfig[1].LowLimit  :=  0.0;
//Ch-2 Configuration
arChannelConfig[2].HighLimit := 5.0;
arChannelConfig[2].LowLimit  :=  -5.0;
```

Program:

Program:

```
fbIH_S20AO4 (
    Enable          := bEnable,
    InOperation=>   bInOperation,
    Error=>         bError,
    ErrorID=>       ErrorID,
    ErrorIdent=>    ErrorIdent,
    BusType         := enBusType,
    BusConfig       := stBusConfig,
    Slot            := uiSlot,
    ChannelConfig   := arChannelConfig,
    OutProcData     := arOutProcData,
    DiagCode        := DiagCode);
```

Outputting analog output values:

```
arChannelConfig[1].NormalizedValue := 10.0;
arChannelConfig[2].NormalizedValue := -5.0;
```

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-AO-4.

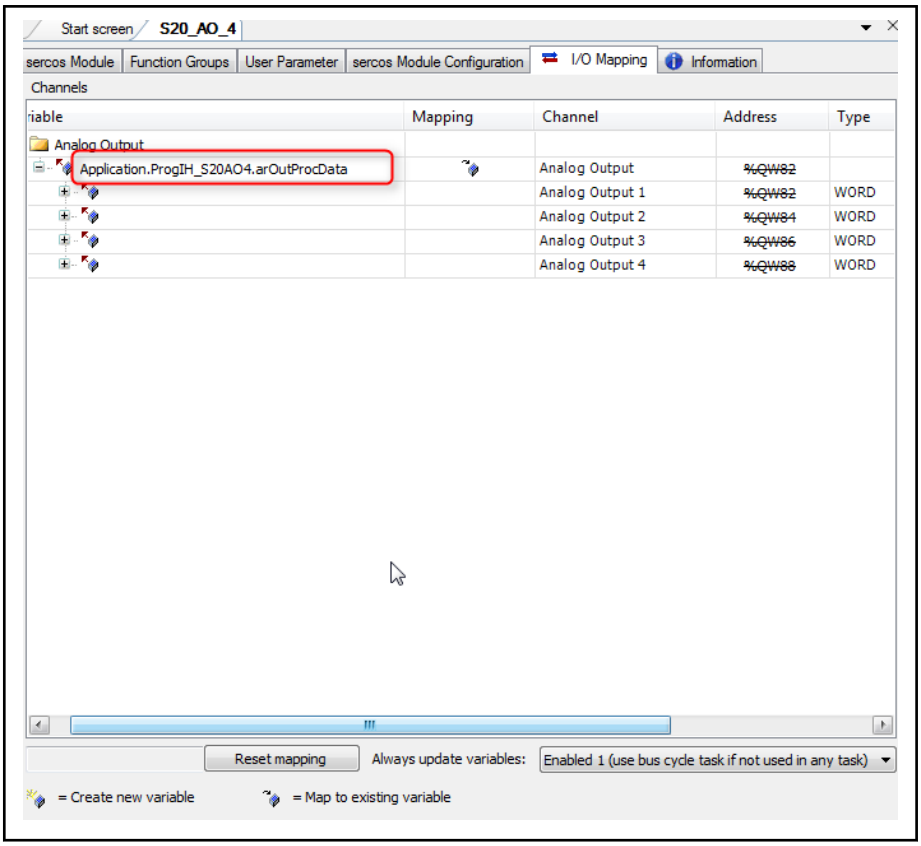


Fig. 23-18: Step 3

23.8 Technology modules (TechnologyIO)

23.8.1 IH_S20SsiAo1

Example program

Example 1: Module S20-SSI-AO-1/1 at the Sercos bus coupler S20-S3-BK+

The example program is used to read the SSI channel and to write analog values from the S20-SSI-AO-1/1 module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for the output ±10V and the SSI channel

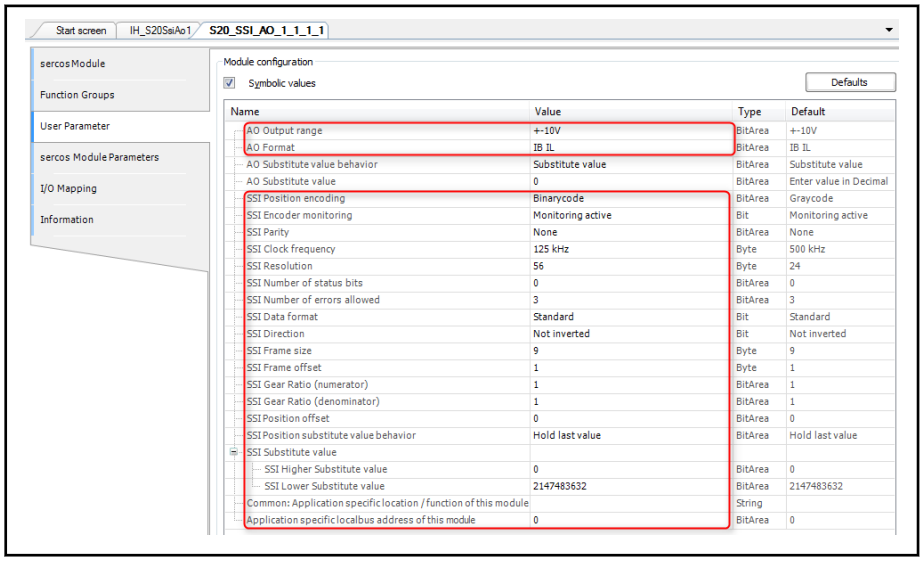


Fig. 23-19: Step 1

Step 2: Variable declaration for the module S20-SSI-AO-1/1 at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20SsiAo1
VAR
    fbIH_S20SsiAo1:     IH_S20SsiAo1; // Function block instance
    bEnable:            BOOL; //TRUE: Function is enabled
    bInOperation:       BOOL; //TRUE: Function Block is working on its actual task
    bError:             BOOL; //TRUE: Indicates an error
    ErrorID:            ERROR_CODE; //Class of error
    ErrorIdent:         ERROR_STRUCT; //Detailed information about error
    enBusType:          IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:        IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:             UINT; //Module slot address. Bus coupler : 0,
                           //1st Module after coupler:1 ... 63rd Module after coupler: 63
    bSetZero:           BOOL; // Resetting position to Zero
    stAOChannelConfig: IH_S20_AO_CONFIG; // Structure for AO channel configuration
    arInProcData:       ARRAY [1..6] OF WORD; // Input process variable
    arOutProcData:      ARRAY [1..6] OF WORD; // Output process variable
    stSSIOutData:       IH_S20_SSI_DATA; // Structure for SSI output status and position data
    stAODiagCode:       IH_S20_DIAG_CODE; // Enumeration for handling AO diagnostics
    wDiagIn:            WORD; // SSI status word on sercos
```

```
    arSSIPosition:     ARRAY [1..4] OF WORD; // SSI position on sercos
    wAnalogOut:        WORD;  // Analog control on sercos
    byDiagOut:         BYTE; // SSI control word on sercos
    rEnc_Position:     REAL; // Encoder actual position
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the technology module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position

// AO Channel configuration
stAoConfig.HighLimit         := 10.0;
stAoConfig.LowLimit          := 0.0;
```

Program:

Program:

```
fbIH_S20SsiAo1 (
    Enable         := bEnable,
    InOperation=>  bInOperation,
    Error=>        bError,
    ErrorID=>      ErrorID,
    ErrorIdent=>   ErrorIdent,
    BusType        := enBusType,
    BusConfig      := stBusConfig,
    Slot           := uiSlot,
    SetZero        := bSetZero,
    AOChannelConfig:= stAOChannelConfig,
    InProcData     := arInProcData,
    OutProcData    := arOutProcData,
    SSIOutData=>   stSSIOutData,
    AODiagCode=>   stAODiagCode);

IF enBusType = IH_S20_BUSTYPE.SERCOS THEN // Process data mapping function on sercos
IH_S20SsiAo1DataMap(
              DiagIn       := wDiagIn,
              SSIPosition  := arSSIPosition,
              OutProcData  := arOutProcData,
              InProcData=> arInProcData,
              AnalogOut=>  wAnalogOut,
              DiagOut=>    byDiagOut);
END_IF
```

☞ Special feature at Sercos to swap process data. The function **IH_S20SsiAo1DataMap** is used for support.

After successful processing, the return value of the function **IH_S20SsiAo1DataMap** is "TRUE".

Reading the input values and writing the output values

```
stAoConfig.NormalizedValue := 5.0; // Normalized analog output value

rEnc_Position := SSIOutData.ActPos; // Encoder actual position
```

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-SSI-AO-1/1.

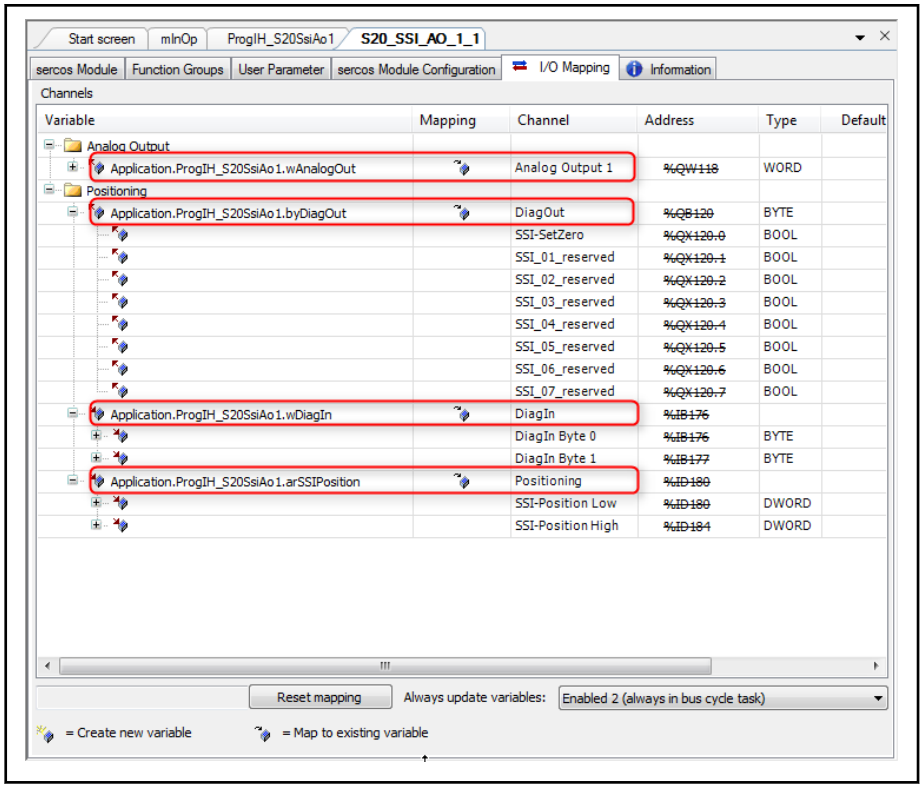


Fig. 23-20: Example 1, step 3

Example 2: Module S20-SSI-AO-1/1 at the Profinet bus coupler S20-PN-BK+

The example program is used to read the SSI channel and to write analog values from the S20-SSI-AO-1/1 module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for the output ±10V and the SSI channel

It is configured as shown in the example of the Sercos bus coupler.

Step 2: Variable declaration for the module S20-SSI-AO-1/1 at the Profinet bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20SsiAo1
VAR
    fbIH_S20SsiAo1:     IH_S20SsiAo1; // Function block instance
    bEnable:            BOOL; //TRUE: Function is enabled
    bInOperation:       BOOL; //TRUE: Function Block is working on its actual task
    bError:             BOOL; //TRUE: Indicates an error
    ErrorID:            ERROR_CODE; //Class of error
    ErrorIdent:         ERROR_STRUCT; //Detailed information about error
    enBusType:          IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:        IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:             UINT; //Module slot address. Bus coupler : 0,
                           //1st Module after coupler:1 ... 63rd Module after coupler: 63
    bSetZero:           BOOL; // Resetting position to Zero
    stAOChannelConfig: IH_S20_AO_CONFIG; // Structure for AO channel configuration
```

```
    arInProcData:       ARRAY [1..6] OF WORD; // Input process variable
    arOutProcData:      ARRAY [1..6] OF WORD; // Output process variable
    stSSIOutData:       IH_S20_SSI_DATA; // Structure for SSI output status and position data
    stAODiagCode:       IH_S20_DIAG_CODE; // Enumeration for handling AO diagnostics
END_VAR
```

Variable initialization for Profinet bus couplers:

Initialization of variables of the function block for the technology module connected to the Profinet bus coupler with the station name "s20-pn-bk" and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                       := IH_S20_BUSTYPE.PROFINET // Connected fieldbus coupler
stBusConfig.Profinet.BusMaster  := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig. Profinet.IdentStr  := 's20-pn-bk';  //Buscoupler address
uiSlot                          := 1; // Module position

// AO Channel configuration
stAoConfig.HighLimit            := 10.0;
stAoConfig.LowLimit             := 0.0;
```

Program:

Program:

```
fbIH_S20SsiAo1 (
    Enable           := bEnable,
    InOperation=>    bInOperation,
    Error=>          bError,
    ErrorID=>        ErrorID,
    ErrorIdent=>     ErrorIdent,
    BusType          := enBusType,
    BusConfig        := stBusConfig,
    Slot             := uiSlot,
    SetZero          := bSetZero,
    AOChannelConfig  := stAOChannelConfig,
    InProcData       := arInProcData,
    OutProcData      := arOutProcData,
    SSIOutData=>     stSSIOutData,
    AODiagCode=>     stAODiagCode);
```

Reading the input values and writing the output values

```
stAoConfig.NormalizedValue := 5.0; // Normalized analog output value

rEnc_Position := SSIOutData.ActPos; // Encoder actual position
```

Linking process data in the I/O mapping at the module in IndraWorks:

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-SSI-AO-1/1.

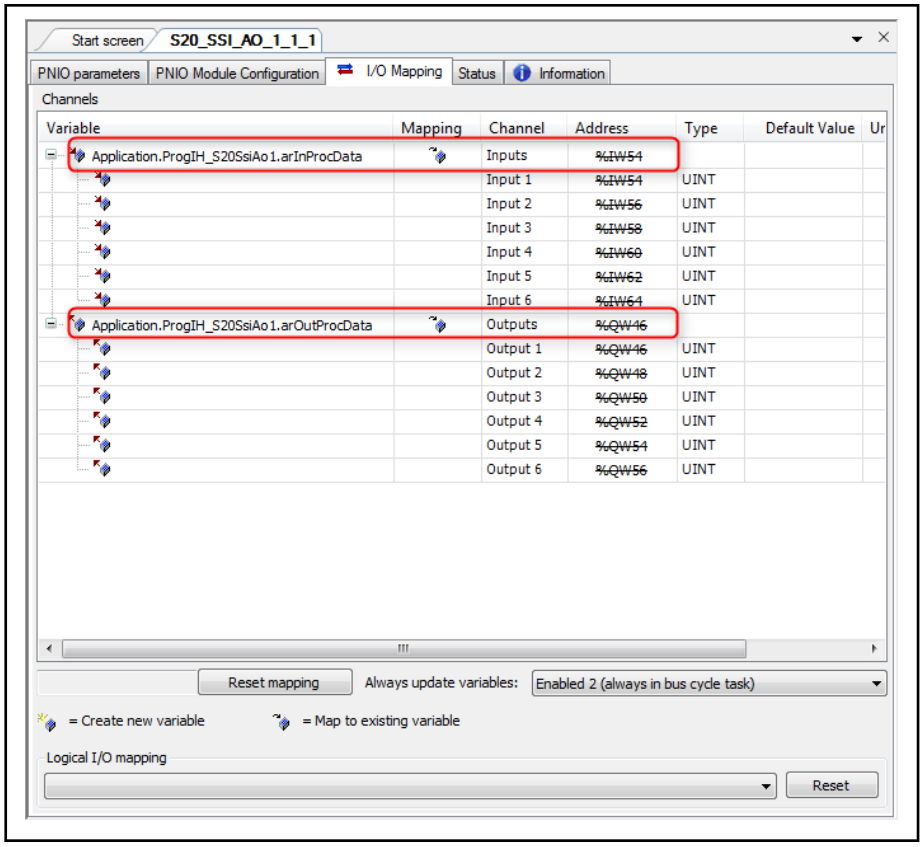


Fig. 23-21: Example 2, step 2

23.8.2 IH_S20CntInc

Example program

Example 1: Module S20-CNT-INC-2/2 at the Sercos bus coupler S20-S3-BK+

The example program is used to read the counter value of channel 1 of the S20-CNT-INC-2/2 module with a cyclic value between 0 and 5000.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for the counter channel 1

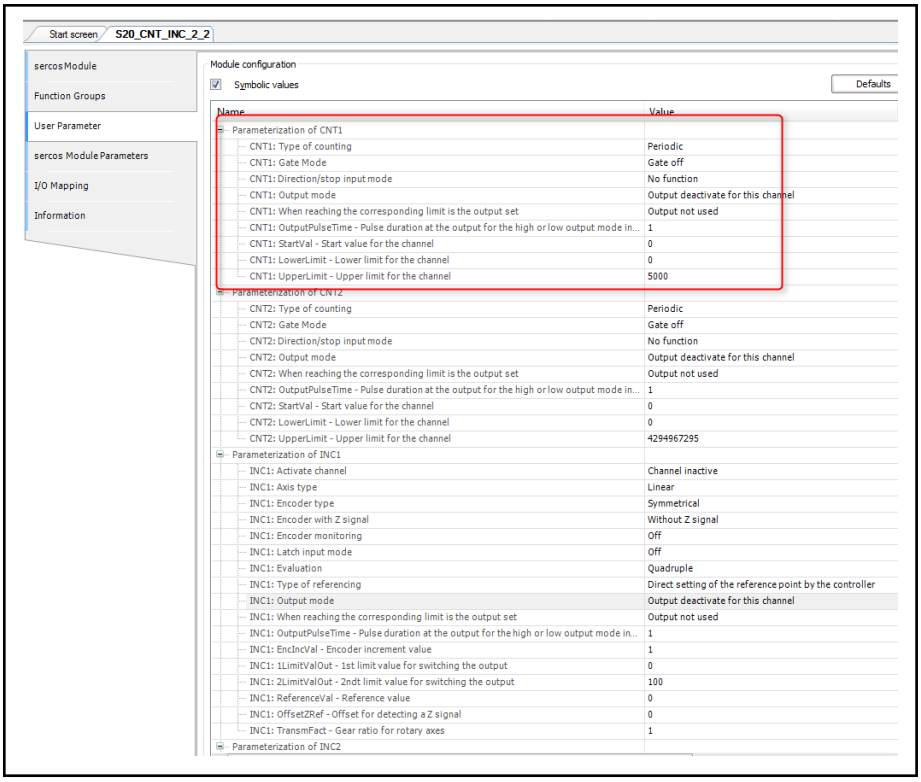


Fig. 23-22: Example 1, step 1

Step 2: Variable declaration for the module S20-CNT-INC-2/2 at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20CntInc
VAR
    fbIH_S20CntInc:      IH_S20CntInc; // FB Instance
    bEnable:             BOOL; //TRUE: Function is enabled
    bInOperation:        BOOL; //TRUE: Function Block is working on its actual task
    bError:              BOOL; //TRUE: Indicates an error
    ErrorID:             ERROR_CODE; //Class of error
    ErrorIdent:          ERROR_STRUCT; //Detailed information about error
    enBusType:           IH_S20_BUSTYPE := IH_S20_BUSTYPE.SERCOS; //Bus type selection
    stBusConfig:         IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:              UINT; //Module slot address. Bus coupler : 0,
                            //1st Module after coupler:1 ... 63rd Module after coupler: 63
    byChannelSelect:     BYTE; // Channel selection
    arCntConfig:         ARRAY[1..2] OF IH_S20_CNT_CONFIG; //Structure for Counter configuration
    arIncConfig:         ARRAY[1..2] OF IH_S20_INC_CONFIG; //Structure for incremental Encoder configuration
    arInProcData:        ARRAY[1..14] OF WORD; //Input Process variable
    arOutProcData:       ARRAY[1..14] OF WORD; //Output Process variable
    bResetDiag:          BOOL; //It deletes the diagnostics memory
                            //and acknowledge the diagnostic messages in the module
    bBusy:               BOOL; //Busy Signal used for PDI handling in sequence
    arCntOutputData:     ARRAY[1..2] OF IH_S20_CNT_DATA; //Structure for counter Output Data
    arIncOutputData:     ARRAY[1..2] OF IH_S20_INC_DATA; //Structure for incremental encoder Output Data
    bCntStart:           BOOL; //TRUE- Start counting
    bCntStop:            BOOL; //TRUE-Stop counting
    bCntDir:             BOOL; //TRUE- Count direction negative
    bCntReset:           BOOL; //TRUE- Count reset
    wCntDiagIn:          WORD; // Cnt Status
    arCntCounterIn:      ARRAY[1..2] OF DWORD; //Counter values
      wIncDiagIn:        WORD; // Inc Status
      arIncPosition:     ARRAY[1..4] OF DWORD; // INC position
      wCntDiagOut:       WORD; //CNT control
      arCntCounterOut:   ARRAY[1..4] OF DWORD; // Counter limits
      wIncDiagOut:       WORD; //INC control
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the technology module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot                       := 1; // Module position
byChannelSelect              := 2#0000_0001; //Channel-1 is selected
```

Program:

Program:

```
fbIH_S20CntInc (
    Enable:= bEnable,
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    Slot:= uiSlot,
    ChannelSelect:= byChannelSelect,
    CntConfig:= arCntConfig,
    IncConfig:= arIncConfig,
    InProcData:= arInProcData,
    OutProcData:= arOutProcData,
    ResetDiag:= bResetDiag,
    CntOutputData=> arCntOutputData,
    IncOutputData=> arIncOutputData);

IF enBusType = IH_S20_BUSTYPE.SERCOS THEN //Process data mapping function on Sercos
    IH_S20CntIncDataMap(
    CntDiagIn:= wCntDiagIn,
    CntCounterIn:= arCntCounterIn,
    IncDiagIn:= wIncDiagIn,
    IncPosition:= arIncPosition,
    OutProcData:= arOutProcData,
    InProcData=> arInProcData,
    IncDiagOut=> wIncDiagOut,
    CntDiagOut=> wCntDiagOut,
    CntCounterOut=> arCntCounterOut);
 END_IF
```

Operating, commanding the counter:

Program:

```
arCntConfig[1].CntStart    := bCntStart; //Start counting
arCntConfig[1].CntStop     := bCntStop; //Stop counting
arCntConfig[1].CntDir      := bCntDir; // Count direction control
arCntConfig[1].CntReset    := bCntReset; // Count reset
```

☞ Special feature at Sercos to swap process data. The function **IH_S20CntIncDataMap** is used for support.

After successful processing, the return value of the function **IH_S20CntIncDataMap** is "TRUE".

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-CNT-INC-2/2.

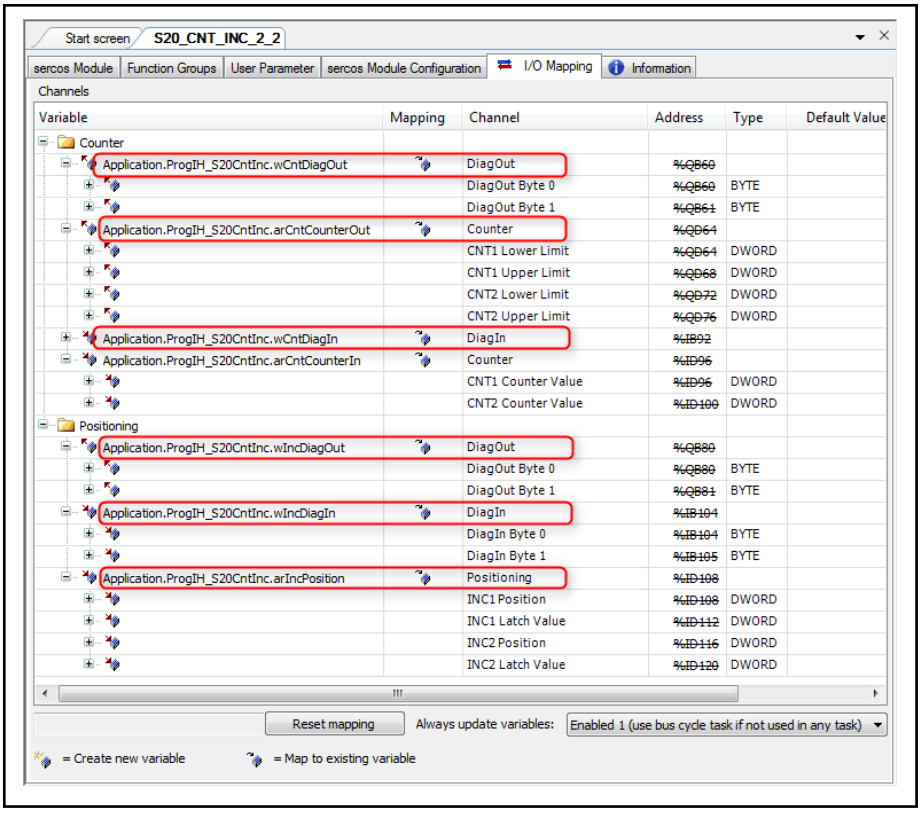


Fig. 23-23: Example 1, step 3

Example 2: Module S20-CNT-INC-2/2 at the Profinet bus coupler S20-PN-BK +

The example program is used to read the counter value of channel 1 of the S20-CNT-INC-2/2 module with a cyclic value between 0 and 5000.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 0: Module configuration in IndraWorks Engineering for the counter channel 1, analog to the Sercos example 1

Step 1: Variable declaration for the module S20-CNT-INC-2/2 at the Profinet bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20CntInc
VAR
    fbIH_S20CntInc:   IH_S20CntInc; // FB Instance
    bEnable:          BOOL; //TRUE: Function is enabled
    bInOperation:     BOOL; //TRUE: Function Block is working on its actual task
    bError:           BOOL; //TRUE: Indicates an error
    ErrorID:          ERROR_CODE; //Class of error
    ErrorIdent:       ERROR_STRUCT; //Detailed information about error
    enBusType:        IH_S20_BUSTYPE := IH_S20_BUSTYPE.SERCOS; //Bus type selection
    stBusConfig:      IH_S20_BUS_CONFIG; //Bus configuration parameters
    uiSlot:           UINT; //Module slot address. Bus coupler : 0,
                        //1st Module after coupler:1 ... 63rd Module after coupler: 63
    byChannelSelect:  BYTE; // Channel selection
```

```
    arCntConfig:      ARRAY[1..2] OF IH_S20_CNT_CONFIG; //Structure for Counter configuration
    arIncConfig:      ARRAY[1..2] OF IH_S20_INC_CONFIG; //Structure for incremental Encoder configuration
    arInProcData:     ARRAY[1..14] OF WORD; //Input Process variable
    arOutProcData:    ARRAY[1..14] OF WORD; //Output Process variable
    bResetDiag:       BOOL; //It deletes the diagnostics memory
                         //and acknowledge the diagnostic messages in the module
    bBusy:            BOOL; //Busy Signal used for PDI handling in sequence
    arCntOutputData:  ARRAY[1..2] OF IH_S20_CNT_DATA; //Structure for counter Output Data
    arIncOutputData:  ARRAY[1..2] OF IH_S20_INC_DATA; //Structure for incremental encoder Output Data
    bCntStart:        BOOL; //TRUE- Start counting
    bCntStop:         BOOL; //TRUE-Stop counting
    bCntDir:          BOOL; //TRUE- Count direction negative
    bCntReset:        BOOL; //TRUE- Count reset

END_VAR
```

Initialization of variables of the function block for the technology module connected to the Profinet bus coupler with the station name "s20-pn-bk" and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                       := IH_S20_BUSTYPE.PROFINET; // Connected fieldbus coupler
stBusConfig.Profinet.BusMaster  := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig. Profinet.IdentStr  := 's20-pn-bk';  //Buscoupler address
uiSlot                          := 1; // Module position
byChannelSelect                 := 2#0000_0001; //Channel-1 is selected
```

Program:

Program:

```
fbIH_S20CntInc (
    Enable:= bEnable,
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    Slot:= uiSlot,
    ChannelSelect:= byChannelSelect,
    CntConfig:= arCntConfig,
    IncConfig:= arIncConfig,
    InProcData:= arInProcData,
    OutProcData:= arOutProcData,
    ResetDiag:= bResetDiag,
    CntOutputData=> arCntOutputData,
    IncOutputData=> arIncOutputData);
```

Operating, commanding the counter:

Program:

```
arCntConfig[1].CntStart      := bCntStart; //Start counting
arCntConfig[1].CntStop       := bCntStop; //Stop counting
arCntConfig[1].CntDir        := bCntDir; // Count direction control
arCntConfig[1].CntReset      := bCntReset; // Count reset
```

Step 4: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-CNT-INC-2/2.

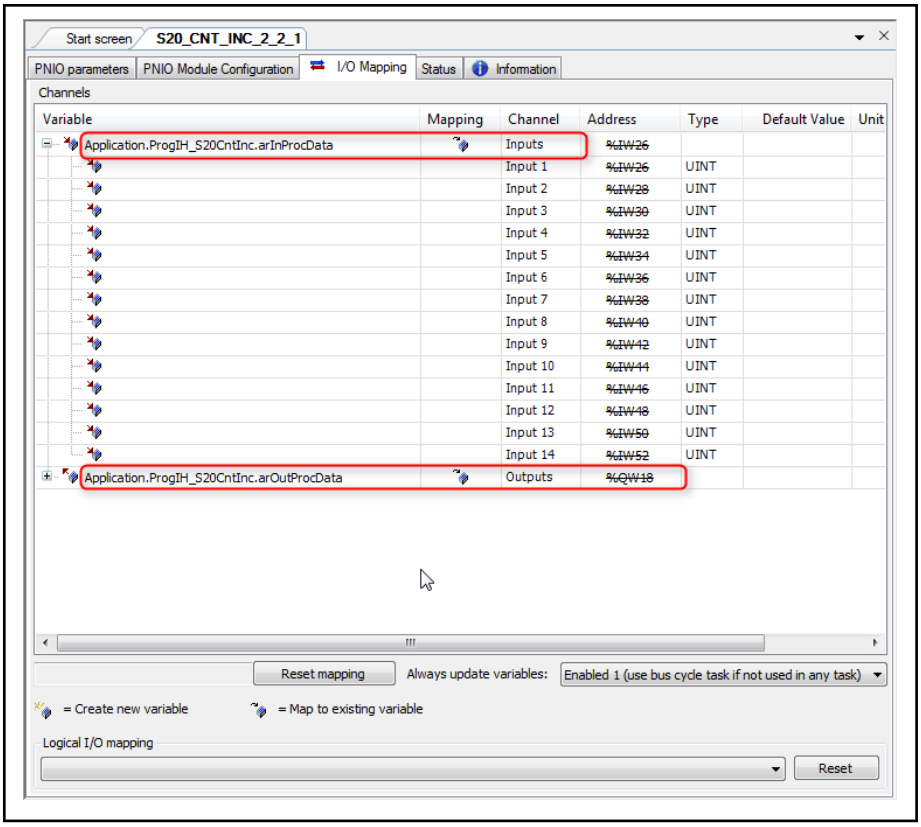


Fig. 23-24: Example 2, step 4

23.8.3 IH_S20RSCom

Example program IH_S20RSCom

Example 1: Module IH_S20RSCom at the Sercos bus coupler S20-S3-BK+

The example program is used for the serial communication of the S20-RS-UNI module.

The field bus used is defined at the input "BusType" using the variable "enBusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for the RS Uni

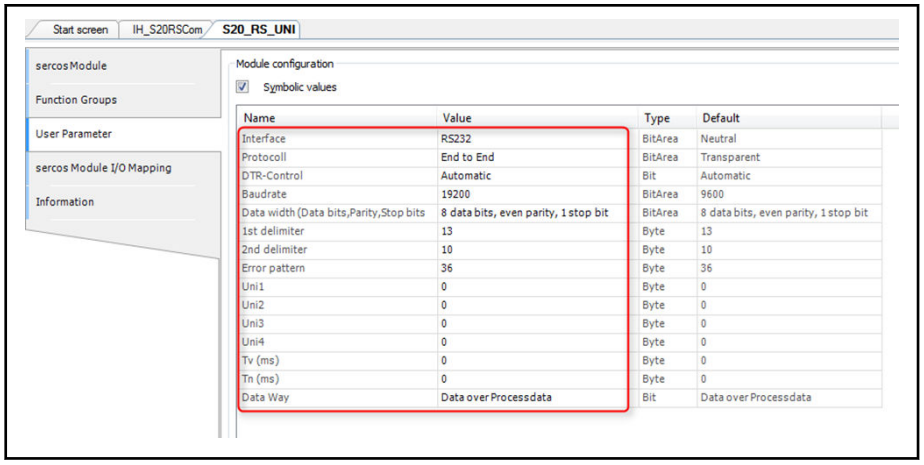


Fig. 23-25: Example 1, step 1

Step 2: Variable declaration for the module IH_S20RSCom at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20_RSCom
VAR
    fbIH_S20RSCom        : IH_S20RSCom; // Function block instance
    bEnable         : BOOL; // TRUE: Function is enabled
    bDone         : BOOL; // TRUE: Function Block has completed its task
    bActive         : BOOL; // TRUE: Function Block is working on its actual task
    bError         : BOOL; // TRUE: Indicates an error
    ErrorID         : ERROR_CODE; // Class of error
    ErrorIdent     : ERROR_STRUCT; // Detailed information about error
    enBusType     : IH_S20_BUSTYPE; // Bus type selection
    stBusConfig     : IH_S20_BUS_CONFIG; // Bus configuration parameters
      uiSlot         : UINT; // Module slot address. Bus coupler : 0,
                  // 1st Mod. after coupler:1 ... 63rd Mod. after coupler: 63
    bSend         : BOOL;
    arSendData         : ARRAY[0..339] OF BYTE;
    uiSizeOfSendData     : UINT;
    bRecv             : BOOL;
    arRecvData         : ARRAY[0..4095] OF BYTE;
    uiSizeOfRecvData     : UINT;
    arInProcData         : ARRAY[0..19] OF BYTE;
    arOutProcData         : ARRAY[0..19] OF BYTE;
    tTimeOut             : TIME := T#3S;
    enAddDiag             : IH_S20_RS_DIAG;
    bRecvBufferNotEmpty     : BOOL;
    uiRecvBufferSize     : UINT;
    uiRecvCounter         : UINT;
END_VAR
```

Variable initialization for Sercos bus couplers:

Initialization of variables of the function block for the technology module connected to the Sercos bus coupler with the Sercos address 69 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType                    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster:= IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr:= 69; //Buscoupler address
uiSlot                       := 1; // Module position
```

Program:

Program:

```
fbIH_S20RSCom(
        Enable:= bEnable,
              Done=> bDone,
              Active=> bActive,
              Error=> bError,
              ErrorID=> ErrorID,
              ErrorIdent=> ErrorIdent,
              BusType:= enBusType,
              BusConfig:= stBusConfig,
              Slot:= uiSlot,
              Send:= bSend,
              SendDataAdr:= ADR(arSendData),
              SizeOfSendData:= uiSizeOfSendData,
              Recv:= bRecv,
              RecvDataAdr:= ADR(arRecvData),
              SizeOfRecvData:= uiSizeOfRecvData,
              InProcDataAdr:= ADR(arInProcData),
              SizeOfInProcData:= SIZEOF(arInProcData),
              OutProcDataAdr:= ADR(arOutProcData),
              SizeOfOutProcData:= SIZEOF(arOutProcData),
              TimeOut:= tTimeOut,
              AddDiag=> enAddDiag,
              RecvBufferNotEmpty=> bRecvBufferNotEmpty,
              RecvBufferSize=> uiRecvBufferSize,
              RecvCounter=> uiRecvCounter);
```

Sending and receiving data:

bSend := TRUE; // To send the data

bRecv := TRUE; // To receive the data

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-RS-UNI.

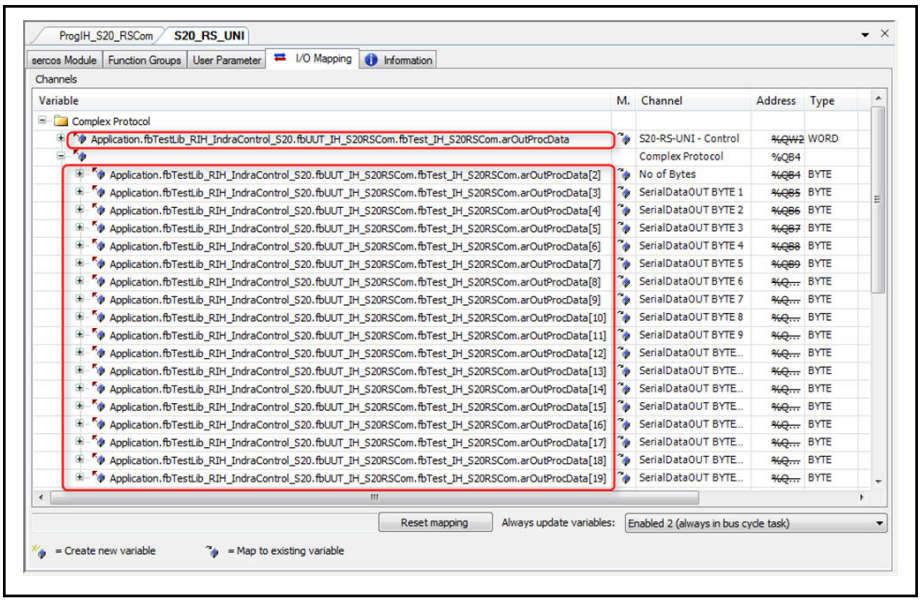


Fig. 23-26: Example 1, step 3, output data

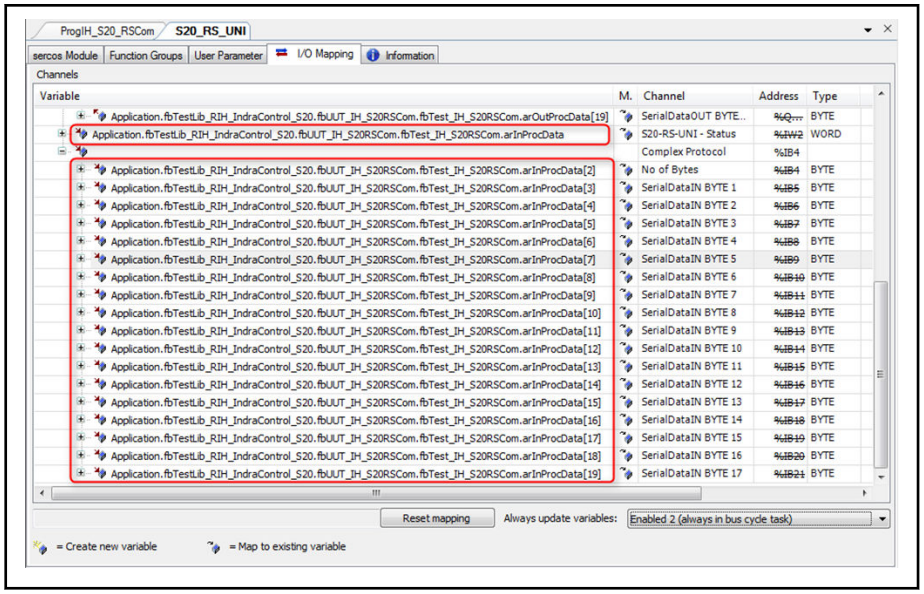


Fig. 23-27: Example 1, step 3, input data

Example 2: Module IH_S20RSCom at the Profibus bus coupler S20-PB-BK

The example program is used for the serial communication of the S20-RS-UNI module.

The field bus used is defined at the input "BusType" using the variable "en-BusType". The address or the name of the bus coupler is defined with "stBusConfig". The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for the RS Uni

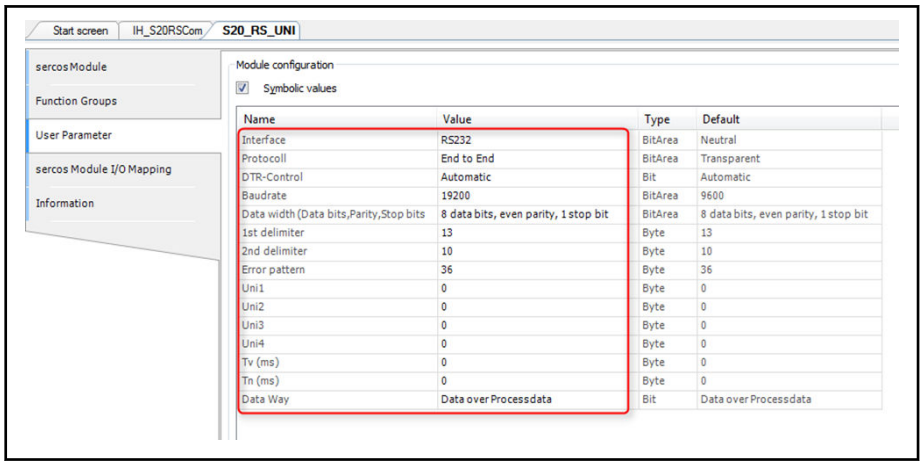


Fig. 23-28: Example 2, step 1

Step 2: Variable declaration for the module IH_S20RSCom at the Profibus bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20_RSCom
VAR
fbIH_S20RSCom    : IH_S20RSCom; // Function block instance
```

```
 bEnable        : BOOL; // TRUE: Function is enabled
    bDone            : BOOL; // TRUE: Function Block has completed its task
    bActive            : BOOL; // TRUE: Function Block is working on its actual task
    bError           : BOOL; // TRUE: Indicates an error
    ErrorID            : ERROR_CODE; // Class of error
    ErrorIdent       : ERROR_STRUCT; // Detailed information about error
    enBusType        : IH_S20_BUSTYPE; // Bus type selection
    stBusConfig        : IH_S20_BUS_CONFIG; // Bus configuration parameters
    uiSlot           : UINT; // Module slot address. Bus coupler : 0,
                  // 1st Module after coupler:1 ... 63rd Module after coupler: 63
    bSend            : BOOL;
    arSendData       : ARRAY[0..339] OF BYTE;
    uiSizeOfSendData    : UINT;
    bRecv            : BOOL;
    arRecvData       : ARRAY[0..4095] OF BYTE;
    uiSizeOfRecvData    : UINT;
    arInProcData       : ARRAY[0..19] OF BYTE;
    arOutProcData        : ARRAY[0..19] OF BYTE;
    tTimeOut        : TIME := T#3S;
    enAddDiag        : IH_S20_RS_DIAG;
    bRecvBufferNotEmpty    : BOOL;
    uiRecvBufferSize    : UINT;
    uiRecvCounter        : UINT;
END_VAR
```

Variable initialization for Profibus bus couplers:

Initialization of variables of the function block for the technology module connected to the Profibus bus coupler with the Profibus address 3 and the module at slot 1 in the program.

Variable initialization:

Program:

```
enBusType            := IH_S20_BUSTYPE.PROFIBUS; // Connected fieldbus coupler
stBusConfig.Profibus.Master    := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Profibus.Segment    := 0;
stBusConfig.Profibus.Station    := 3; // Bus coupler node address
uiSlot                  := 1; // Module position
```

Program:

Program:

```
fbIH_S20RSCom(
    Enable:= bEnable,
    Done=> bDone,
    Active=> bActive,
    Error=> bError,
    ErrorID=> ErrorID,
    ErrorIdent=> ErrorIdent,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    Slot:= uiSlot,
    Send:= bSend,
    SendDataAdr:= ADR(arSendData),
    SizeOfSendData:= uiSizeOfSendData,
    Recv:= bRecv,
    RecvDataAdr:= ADR(arRecvData),
    SizeOfRecvData:= uiSizeOfRecvData,
    InProcDataAdr:= ADR(arInProcData),
    SizeOfInProcData:= SIZEOF(arInProcData),
    OutProcDataAdr:= ADR(arOutProcData),
    SizeOfOutProcData:= SIZEOF(arOutProcData),
    TimeOut:= tTimeOut,
    AddDiag=> enAddDiag,
    RecvBufferNotEmpty=> bRecvBufferNotEmpty,
    RecvBufferSize=> uiRecvBufferSize,
    RecvCounter=> uiRecvCounter);
```

Sending and receiving data:

bSend := TRUE; // To send the data

bRecv := TRUE; // To receive the data

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Linking process data in the I/O mapping to the variables declared in the PLC program for the module S20-RS-UNI.

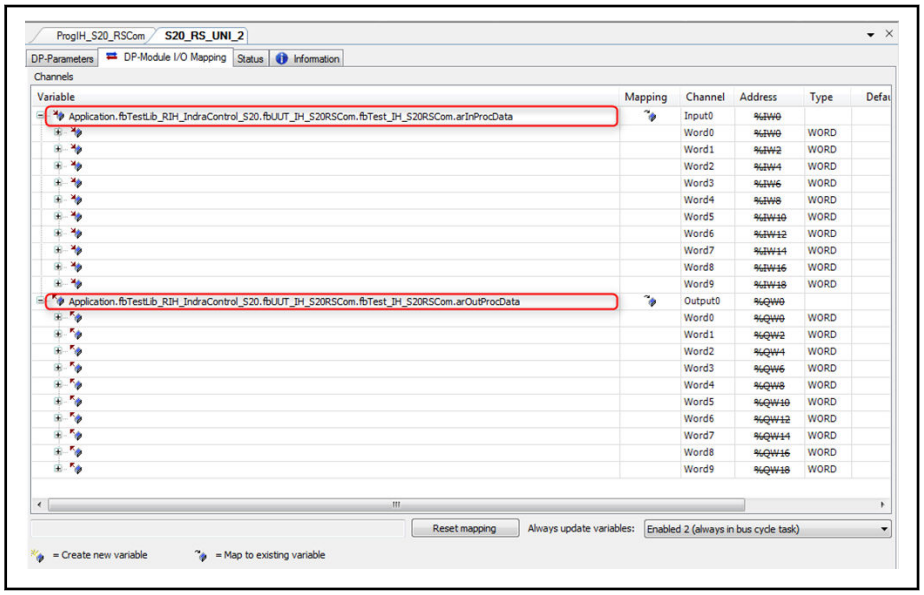


Fig. 23-29: Example 2, step 3

Example programs IH_S20RSComType02

Example 3: Module IH_S20RSComType02 at the Sercos bus coupler S20-S3-BK+

The example program is used for the serial communication of the S20-RS-UNI module.

The function block “IH_S20ComConfiguration” defines the field bus used, address or the bus coupler name. They are transferred using the S20BusHandle. The module slot is defined at the "Slot" input.

TRUE at the "bEnable" input applies the configuration and the function block reads the current configuration of the module.

Step 1: Module configuration in IndraWorks Engineering for the RS Uni

Refer to Example 1

Step 2: Variable declaration for the module IH_S20RSCom at the Sercos bus coupler

Declaration:

Program:

```
PROGRAM ProgIH_S20RSComType02
VAR
    fbIH_S20ComConfiguration: IH_S20ComConfiguration;  // Function block instance
    bEnableComConfig:         BOOL; // Function block processing enabled( TRUE - once, edge-controlled )
    bInOperationComConfig:    BOOL; //TRUE: Function Block is working on its actual task
    bErrorComConfig:          BOOL; //TRUE: Indicates an error
    ErrorIDComConfig:         ERROR_CODE; //Class of error
    ErrorIdentComConfig:      ERROR_STRUCT; //Detailed information about error
    enBusType:                IH_S20_BUSTYPE; //Bus type selection
    stBusConfig:              IH_S20_BUS_CONFIG; //Bus configuration parameters
    dwS20BusHandle:           DWORD; //Bus handle of S20IO configuration

    fbIH_S20RSComType02:      IH_S20RSComType02; // Function block instance
 bEnable:                  BOOL; // TRUE: Function is enabled
    bDone:                    BOOL; // TRUE: Function Block has completed its task
    bActive:                  BOOL; // TRUE: Function Block is working on its actual task
    bError:                   BOOL; // TRUE: Indicates an error
    ErrorID:                  ERROR_CODE; // Class of error
    ErrorIdent:               ERROR_STRUCT; // Detailed information about error
    stRSConfig:               IH_S20_RS_CONFIG;
    bSend:                    BOOL;
```

```
 arSendData:               ARRAY[0..339] OF BYTE;
 uiSizeOfSendData:         UINT;
 bRecv:                    BOOL;
 arRecvData:               ARRAY[0..4095] OF BYTE;
 uiSizeOfRecvData:         UINT;
 arInProcData:             ARRAY[0..19] OF BYTE;
 arOutProcData:            ARRAY[0..19] OF BYTE;
 tTimeOut:                 TIME := T#3S;
    enAddDiag:                IH_S20_RS_DIAG;
    bRecvBufferNotEmpty:      BOOL;
    uiRecvBufferSize:         UINT;
    uiRecvCounter:            UINT;

END_VAR
```

Variable initialization and program for Sercos bus coupler:

Program:

```
//Step-1: Variable Initiliasation:

enBusType    := IH_S20_BUSTYPE.SERCOS; // Connected fieldbus coupler
stBusConfig.Sercos.BusMaster := IL_BUSMASTER.IL_BUSMASTER_0;
stBusConfig.Sercos.SercosAdr := 69; //Buscoupler address
uiSlot  := 1; // Module position

// Channel configuration
stRSConfig.InterfaceType     := IH_S20_RS_INTERFACE.RS232;
stRSConfig.Protocol          := IH_S20_RS_PROTOCOL.END_TO_END;
stRSConfig.Baudrate          := IH_S20_RS_BAUDRATE.BAUDRATE_19200;
stRSConfig.DataWidth         := IH_S20_RS_DATAWIDTH.DATABIT8_EVEN_STOPBIT1;
stRSConfig.FirstDelimiter     := 16#0D;
stRSConfig.SecondDelimiter    := 16#0A;
stRSConfig.ErrorPattern     := 16#24;
stRSConfig.Uni1          := 16#00;
stRSConfig.Uni2          := 16#00;
stRSConfig.Uni3          := 16#00;
stRSConfig.Tv              := 16#00;
stRSConfig.Tn              := 16#00;

//Step-2: Program:

fbIH_S20ComConfiguration(
    Enable:= bEnableComConfig,
    InOperation=> bInOperationComConfig,
    Error=> bErrorComConfig,
    ErrorID=> ErrorIDComConfig,
    ErrorIdent=> ErrorIdentComConfig,
    BusType:= enBusType,
    BusConfig:= stBusConfig,
    S20BusHandle=> dwS20BusHandle);

fbIH_S20RSComType02(
        Enable:= bEnable,
              Done=> bDone,
              Active=> bActive,
              Error=> bError,
              ErrorID=> ErrorID,
              ErrorIdent=> ErrorIdent,
        S20BusHandle:= dwS20BusHandle;
              Slot:= uiSlot,
              RSConfig:= stRSConfig,
              Send:= bSend,
             SendDataAdr:= ADR(arSendData),
             SizeOfSendData:= uiSizeOfSendData,
              Recv:= bRecv,
              RecvDataAdr:= ADR(arRecvData),
             SizeOfRecvData:= uiSizeOfRecvData,
              InProcDataAdr:= ADR(arInProcData),
             SizeOfInProcData:= SIZEOF(arInProcData),
              OutProcDataAdr:= ADR(arOutProcData),
              SizeOfOutProcData:= SIZEOF(arOutProcData),
              TimeOut:= tTimeOut,
              AddDiag=> enAddDiag,
              RecvBufferNotEmpty=> bRecvBufferNotEmpty,
              RecvBufferSize=> uiRecvBufferSize,
              RecvCounter=> uiRecvCounter);
```

Sending and receiving data:

bSend := TRUE; // To send the data

bRecv := TRUE; // To receive the data

Step 3: Linking process data in the I/O mapping at the module in IndraWorks

Refer to Example 1, step 3

24 RIH_IndraControl_S20_KNX

24.1 General information

The library "RIH_IndraControl_S20_KNX" includes function blocks allowing the PLC program to access the IndraControl S20 KNX module.

IndraControl XM controls support the library (IndraControl S20 Sercos bus coupler).

Functions

IndraControl XM controls ③ contain a software interface consisting of the " IH_S20KNXClient" and of the function blocks for the individual data point types, refer to the table tab. 24-1 "KNX service and data point types" on page 950.

The function block " IH_S20KNXClient" ③ manages the communication to the object server ⑤ using a communication protocol ④ that is based on the serial FT1.2 protocol.

The S20 KNX-1 module ⑥ provides the KNX object server ⑤ and includes the "ServerItems" and the "DataPoints".

The KNX object server ⑤ is configured and parameterized using a programming tool[1)] ⑨ on the S20-KNX-1 module ⑥.

An external KNX power supply unit ⑧ supplies the S20 KNX-1 module and the connected KNX devices ⑦.

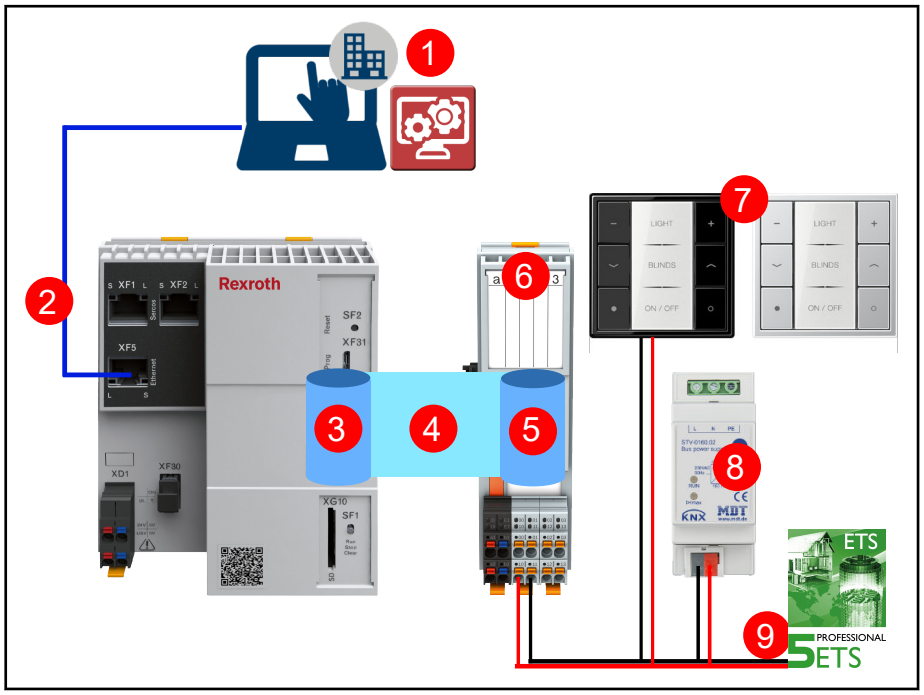


Fig. 24-1: Functional diagram – S20 KNX-1 module and network topology

The library and the S20-KNX-1 module support the following services and data point types:

1) *USB KNX interface, or ETH KNX gateway with interface*

KNX service and data point types

Service	Function block	Description	Data width
IndraControl XM And S20-S3-BK+	IH_S20KNXClient	Communication with the S20 KNX-1 module (requires the S20BusHandle from the function block "IH_S20ComConfiguration")	
DPT1	IH_S20KNXDPT1	Binary	1 bit
DPT2	IH_S20KNXDPT2	Binary with priority	2 bits
DPT3	IH_S20KNXDPT3	Dimming up/down	4 bits
DPT4	IH_S20KNXDPT4	Characters	1 byte
DPT5	IH_S20KNXDPT5	Percentage value	1 byte
DPT6	IH_S20KNXDPT6	Signed value	1 byte
DPT7	IH_S20KNXDPT7	Unsigned value	2 bytes
DPT8	IH_S20KNXDPT8	Signed value	2 bytes
DPT9	IH_S20KNXDPT9	Floating point value	2 bytes
DPT10	IH_S20KNXDPT10	Time	3 bytes
DPT11	IH_S20KNXDPT1	Date	3 bytes
DPT12	IH_S20KNXDPT12	Unsigned value	4 bytes
DPT13	IH_S20KNXDPT13	Signed value	4 bytes
DPT14	IH_S20KNXDPT14	Floating point value	4 bytes
DPT16	IH_S20KNXDPT16	Text/string	14 bytes
DPT18	IH_S20KNXDPT18	Scene with learning function	1 byte

Tab. 24-1:

☞ The module S20 KNX-1 does not support DPT15 and DPT17!

For the documentation on the function blocks of the library RIH_IndraControl_S20_KNX, go to the IndraWorks library manager.

Open the library manager in the application and select the respective function block in the RIH_IndraControl_S20_KNX library.

The "Documentation" folder includes the following function block information:

- Special features
- Structures
- Enumerations
- Program code examples for the different DPT function blocks in the folder **_CodeExamples** (for more examples and commissioning videos, go to the product in the Internet)

☞ The recommended PLC cycle time is 10 ms for the function block IH_S20KNXClient.

Diagnostic and error codes

For the error codes and function blocks, refer to the following table:

☞ For specifications on the error codes of the S20 KNX-1 module, refer to the data sheet of the module.

Potential error sources:

- The KNX (ETS) configuration of the S20 KNX-1 module is invalid
- The KNX voltage is missing or overloaded
- Error in the KNX cabling/installation

Diagnostics and error codes

Additional 1	Additional 2	ErrorID	Description
16#0001	16#00000002	INPUT_RANGE_ERROR	Wrong slot
16#0001	16#00000005	INPUT_RANGE_ERROR	TimeOut beyond the range (2 s..60 min)
16#0002	16#00000001	INPUT_INVALID_ERROR	Wrong module or wrong slot
16#0006	16#00000001	STATE_MACHINE_ERROR	Internal State Machine Error Restart the function block
16#0007	16#00000000	RESOURCE_ERROR	Null Pointer, invalid S20BusHandle
16#0008	16#00000000	DEVICE_ERROR	TimeOut
16#0201	16#00000001	INPUT_RANGE_ERROR	DPT ID beyond the range 1...1,000
	16#00000002	INPUT_RANGE_ERROR	DPT name beyond the range
	16#00000003	INPUT_RANGE_ERROR	DPT value beyond the range. The input range is DPT-specific
	16#00000004	INPUT_RANGE_ERROR	ReadWrite beyond the range
16#0206	16#00000001	DEVICE_ERROR	DPT is disabled in the KNX configuration (ETS)
	16#00000002	DEVICE_ERROR	IH_S20KNXClient not ready (busy) • KNX client initialized • KNX client not enabled
	16#00000003	DEVICE_ERROR	Response telegram cannot be identified
	16#00000004	DEVICE_ERROR	KNX bus not connected
	16#00000005	DEVICE_ERROR	DPT configuration with ETS software (ETS download) changed
	16#xxxx_00yy	DEVICE_ERROR	Negative response for a read/write action • xxxx - Wrong DPT ID `16#0001` - `16#03E8` • 00yy - Error numbers 0-11 Refer to tab. 24-3 "Overview on KNX protocol errors" on page 952

Additional 1	Additional 2	ErrorID	Description
16#0207	16#00000001	RESOURCE_ERROR	DPT type does not match the object server configuration
	16#00000002	RESOURCE_ERROR	Data exchange token to the KNX client is consumed. Up to 260 tasks can be processed simultaneously. Wait for free token and give it a new try
	16#00000003	RESOURCE_ERROR	DPT is not configured in the S20 KNX-1 module
	16#00000004	RESOURCE_ERROR	DPT configuration (flags) do not fulfil the requirement for deep reading (Deep Read) Write, Transmit and Update Flag have to be enabled in the ETS configuration

Tab. 24-2:

Overview on KNX protocol errors

Error number	Description
0	No error
1	Internal error
2	Element not found
3	Buffer too small
4	It cannot be written on the Item
5	Service not supported
6	Wrong service parameter (Bad)
7	Wrong ID (Bad)
8	Wrong command (Bad)/value (value)
9	Incorrect length
10	Message inconsistent
11	Object server busy

Tab. 24-3:

Approach in case of error

Perform the following checks if an error was issued:

1. Check KNX voltage supply
2. Check ETS configuration
3. Check status and error code of the function blocks

DPT Status/Communication Status/Error

- **IH_S20_KNX_STATUS** indicates what the functions block **IH_S20KNXClient** is currently processing or what is currently faulty
- Each of the KNX DPT function blocks is provided with a **Valid** output that signals in the TRUE state, that a KNX data point is valid and that the status corresponds to the status in the KNX bus of the real KNX application.

There can be different reasons if **Valid** is FALSE:

- Initial state, KNX is not yet initialized
- No DPT read or written from the user program, no change in value in the actual KNX application
- Check **IH_S20KNXClient** for pending diagnostics
- S20-KNX-1 module in the error (check the diagnostic code and the at the S20-KNX-1 module)
- KNX voltage missing (is signaled at the S20-KNX-1 module with the error 0x3402 and can be identified due to the E2-LED red)

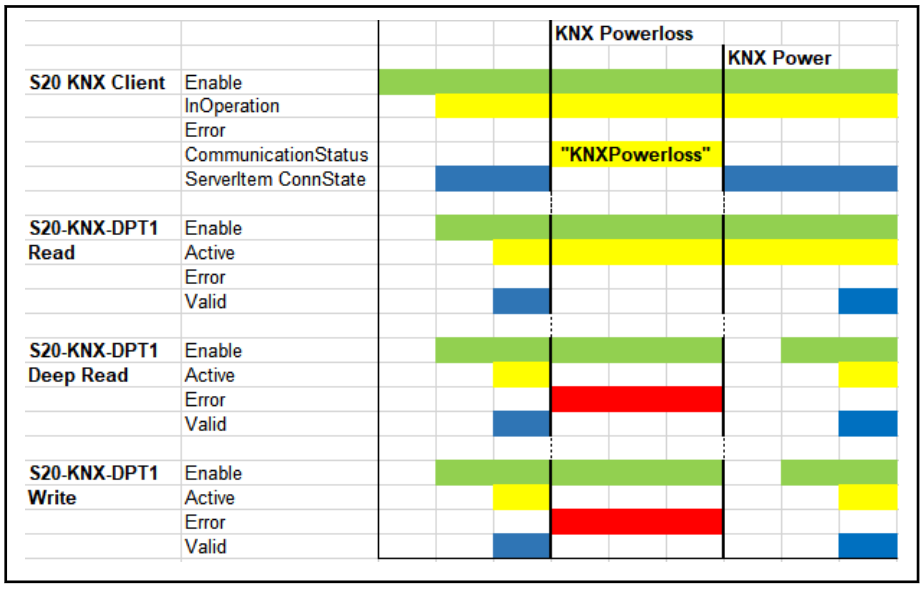


Fig. 24-2: Signal-time diagram of the KNX voltage error response

In the KNX voltage error state, KNX data points can still be read from the buffer of the **IH_S20KNXClient** using the KNX DPT function blocks. Writing and "deep" reading causes an error, as the KNX bus is accessed. If the KNX voltage is present anymore, **Valid** becomes **TRUE** again on the KNX bus after the first update of the respective data point

IH_S20KNXClient

The function block **IH_S20KNXClient** is the software interface between the S20 KNX-1 module and the KNX data point type function blocks (using **KNXHandle**) .

An independent instance of the **IH_S20KNXClient** function block is required for the each configured S20 KNX-1 module.

To call the **IH_S20KNXClient** function block, the **IH_S20ComConfiguration** function block is additionally required. The function block provides the required **S20BusHandle** for the **IH_S20KNXClient** function block.

The function blocks of the individual data types run analogously. The interface is the **KNXHandle** at the **IH_S20KNXClient** function block.

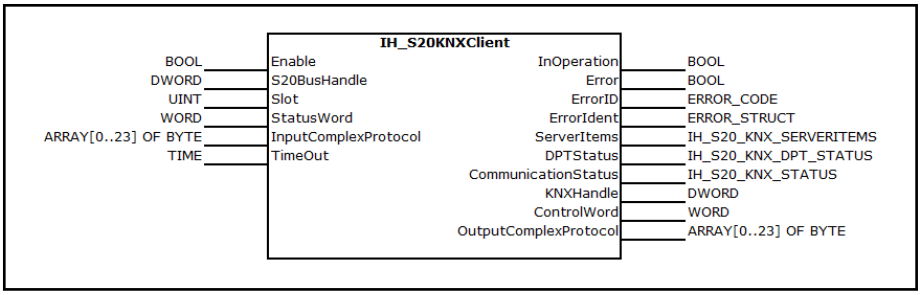


Fig. 24-3: IH_S20KNXClient function block

To set up the communication to the S20 KNX-1 module, set the "Enable" input to `TRUE` at the **IH_S20KNXClient**.

When changing from `FALSE` to `TRUE` at the "Enable" input, the interface is initialized.

All configured data points of the S20 KNX-1 modules are retrieved and stored in an internal data area (up to 1,000 data points).

The initialization is completed after all data points were loaded. The "InOperation" output becomes `TRUE`.

The S20 KNX-1 module is now ready.

The output "CommunicationStatus" indicates the operational readiness with the status "CLIENT_READY".

The initialization can take up to some seconds. It is displayed with "`INIT_******`" at the "CommunicationStatus" output.

☞ It is initialized with each new edge from `FALSE` to `TRUE` at the "Enable" input.

After the initialization, all telegrams of the received individual KNX data points are analyzed at runtime and stored in the RAW data of the function block **IH_S20KNXClient**.

The **KNXHandle** in the application is used to communicate between the function blocks of the different KNX data point types and the **IH_S20KNXClient** function block.

The output parameter "DPTStatus" provides the following RAW data from the KNX configuration to the user:

- DPTID
- DPT type
- Configuration flags of the ETS configuration
- State
  - TransmissionStatus
  - ReadrequestFlag
  - UpdateFlag: Flag is set if ActValue of the respective DPT-ID was updated
  - ValidFlag
- KNX user data (14 bytes)

This information is analyzed and prepared using the function blocks of the individual data point types (output parameter not mandatorily required/optional).

The output parameter "ServerItems" provides the following information on the loaded KNX project to the PLC application:

- Hardware type
- Hardware version
- Firmware version
- KNX Manufacturer Code DEV
- KNX Manufacturer Code APP
- ETS Applikation ID
- ETS application version
- Serial number
- Bus status (KNX)
- Protocol version
- Indication Sending
- Maximum number of data points
- Number of the configured data points (depends on the created ETS project, whether with 250 data points or more)

The available input and output parameters of the function blocks are determined by the included or configured data point types.

IH_S20KNXDPTxx

The individual function blocks IH_S20KNXDPTxx are provided with a "ReadWrite" input with the enumeration "IH_S20_KNX_RW".

The following write/read modes are supported:

IH_S20_KNX_RW

ReadWrite Mode	Description
IDLE	"ActValue" (current value) is read (default mode)
WRITEONCE	"SetValue" is written once. If the "Threshold" input is used additionally, it is only written once when "SetValue" is greater than or equal to the “Threshold” value. Writing can be forced if "ActValue" is equal to "ACTValue" and if "Threshold" is set to zero. Cyclic execution is limited to 100 ms.
WRITEAUTO	"SetValue" is always written when this values differs from its last specification. If the "Threshold" input is used additionally, it is always written when "SetValue" is greater than or equal to the “Threshold” value and if it differs from the previous "SetValue". It is written once after a rising edge at “Enable” if the value differs from "ActValue" and if it is greater than "Threshold". Cyclic writing is limited to 100 ms.
READVIABUS	The KNX bus forces direct reading. It is read actively on the KNX bus

If WRITEONCE, WRITEAUTO and READVIABUS were successful, “Done” is shown for a PLC cycle. The cyclic execution is limited to 100 ms.

If a “Threshold” value is used and if “SetValue” is to be overwritten by "0" or if writing is forced, specify a “Threshold” of 0.

24.2 Application example

A example application is described in the following starting from the KNX configuration on the S20-KNX-1 module up to the application program on the control to read and write a KNX data point.

Prerequisite

1. The S20-KNX-1 is ready, connected to an XM control or S20-S3-BK+ and operating
2. KNX power supply unit and supply present
3. KNX devices connected to a KNX power supply unit and an S20-KNX-1
4. KNX programming device present, e.g. a KNX USB interface or a KNX IP gateway and connected to the S20-KNX-1 and the engineering PC
5. ETS5 software downloaded from the KNX website and installed
6. Required KNX description files can be directly loaded to the ETS5 software if the files are available in the ETS catalog. Otherwise, contact the KNX device manufacturer and import the files directly into the ETS5 software

Creating ETS project

1. Open ETS5 software and create a new project:
   - Specify a name
   - Select backbone "IP"
   - Select the topology "Create Line 1.1" and select "TP"
   - Select the group address style "Three Level"

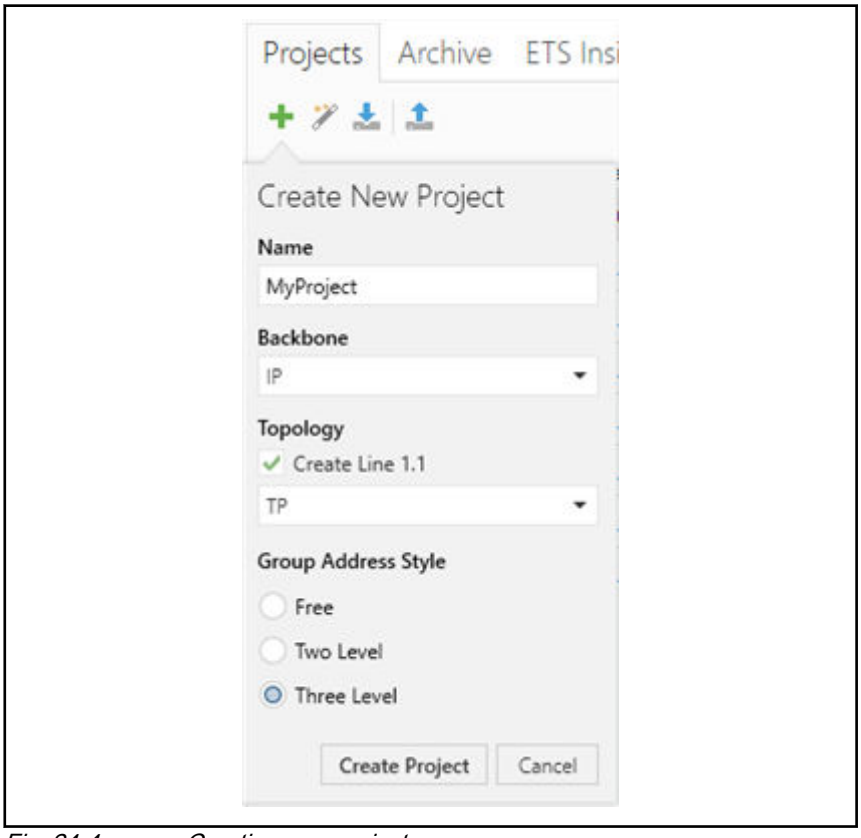


Fig. 24-4: Creating new project

Project view:

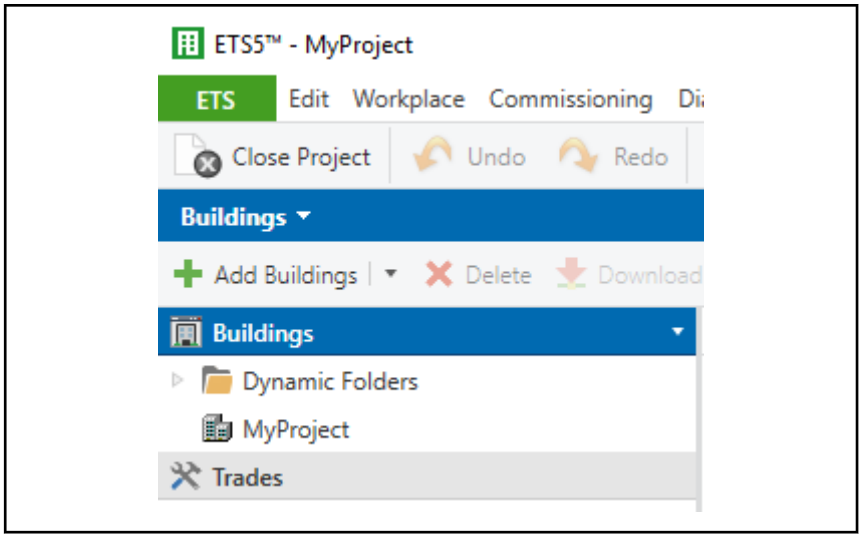


Fig. 24-5: ETS project view

2. Add a room to the project:
   1. Right click on "MyProject" und add the room
   2. Specify and create a room name

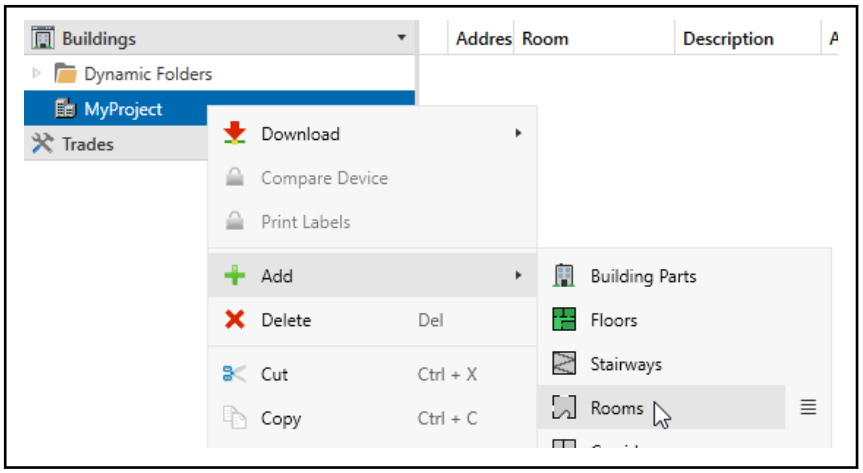


Fig. 24-6: Adding a room

3. Add devices, analogously to the room:

   Right-click on the room and add devices. The device catalog opens automatically. Select the devices in the catalog and add them to the room. Use drag&drop.

   The example shows the module "S20-KNX-1" and the switch/pushbutton "4072.02LED" (Albrecht Jung company) and optionally also the KNX power supply unit MDT STV-320.02

☞ KNX power supply units are part of the configuration and ensure that the KNX bus supply is sufficient upon the validation.

---

4. Configure the S20-KNX-1 module for the room "MyRoom", select the device and the "Parameter" tab. In the **General** menu, select "Enabled" for "Send Indications".

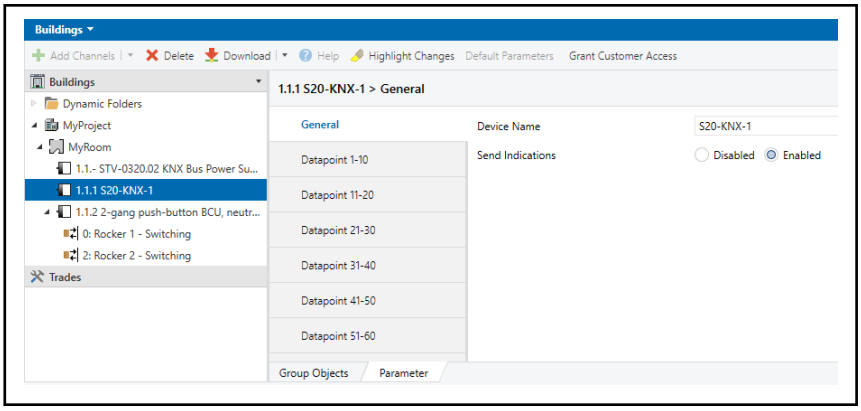


Fig. 24-7: "MyRoom" configuration

5. From the **General** menu, go to **Data point 1-10** and select the type "DPT 1 - Binary-1 Bit" once for "Datapoint type 1" and once for Datapoint type 2

   For a better overview, provide a brief description for each point "Datapoint description".

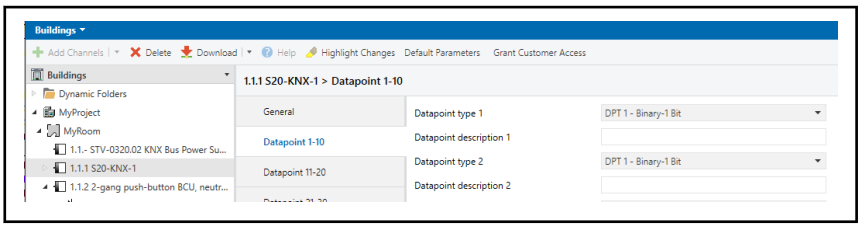


Fig. 24-8: Datapoint 1-10 configuration

6. When changing to the "Group Objects" tab, the two created data points are displayed. The respective flags are set for each data point via "Properties".

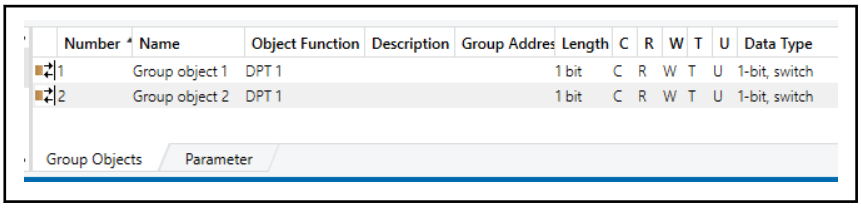


Fig. 24-9: "Group Objects" tab

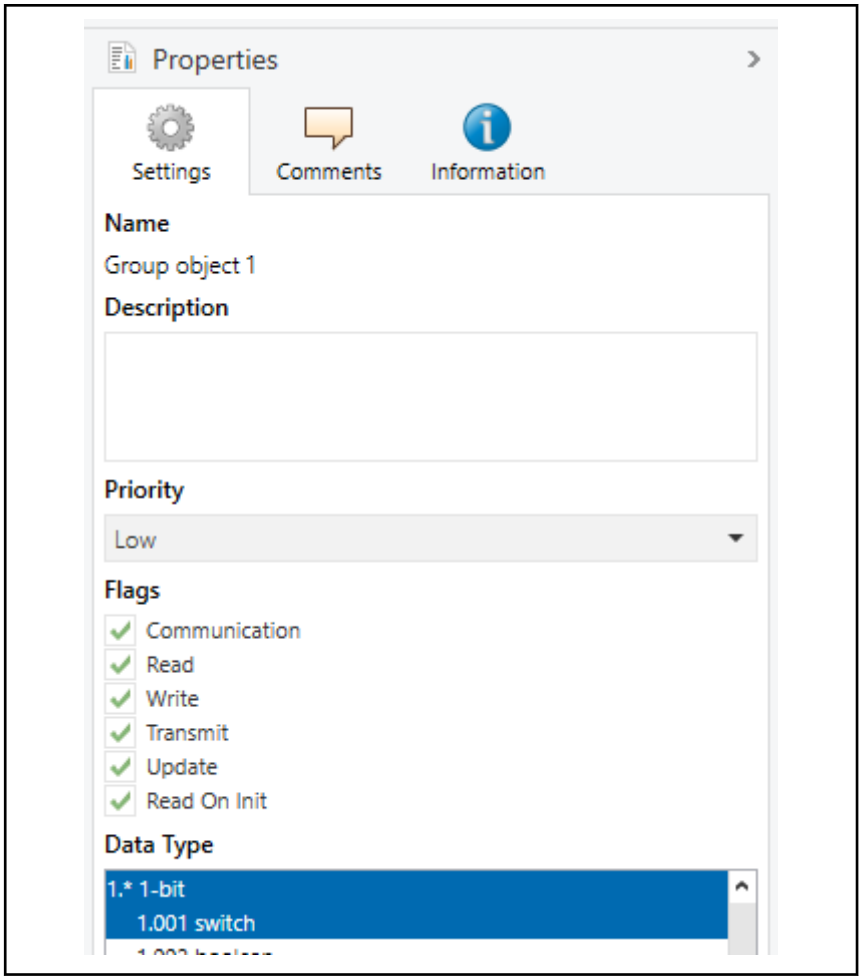


Fig. 24-10: "Properties" tab

7. At the pushbutton, the default settings can remain except for the LED response. In "Function", the behavior of "Button-press display" has to be deselected and "Status indicator (via LED object)" has to be selected. A new data point is automatically inserted at the pushbutton. For this data point, enable the flags under "Properties" as well.

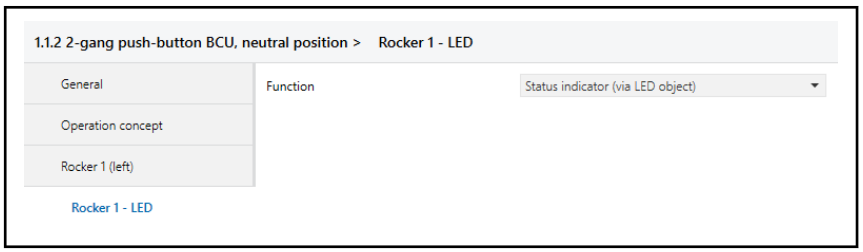


Fig. 24-11: Pushbutton settings

The individual KNX devices are prepared. Use groups to interlink these devices.

8. To add the group view, select **Workplace ▸ Open New Panel ▸ Group addresses** in the menu

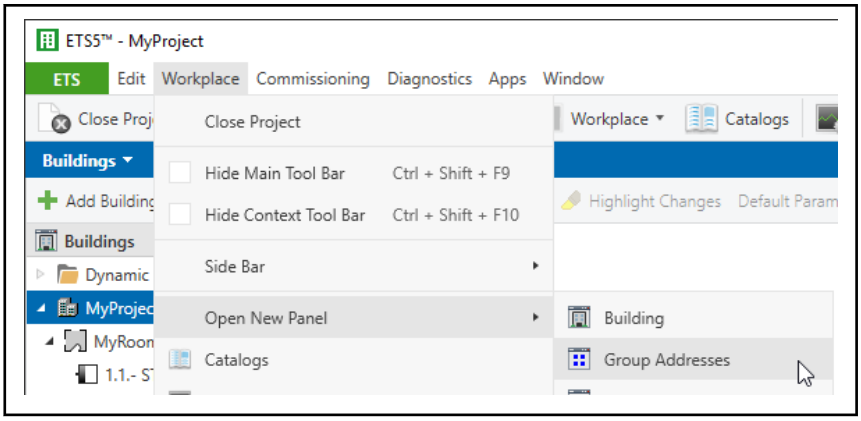


Fig. 24-12: "Workplace" menu

9. Go to **Add Main Groups** to add the groups:
   1. First level "MainGroup"
   2. Next level "MiddleGroup"
   3. Group address "RockerSwitch1"
   4. Group address "RockerLED"

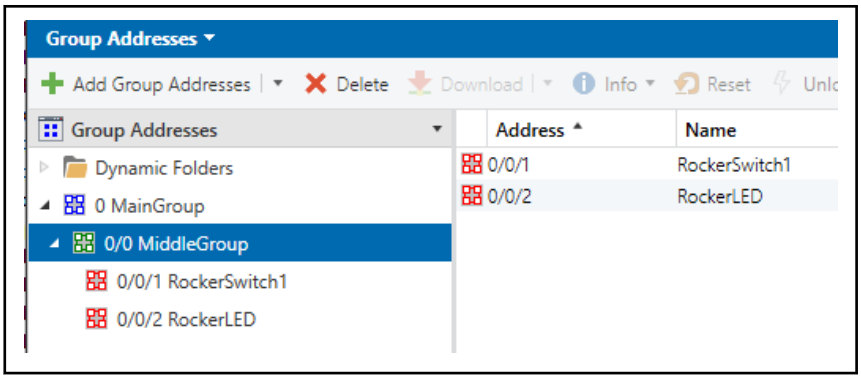


Fig. 24-13: Adding groups

10. Add the data points to the group addresses via drag&drop. Adding them to a group is required for a unique addressing on the KNX. Data points are linked to exchange data for reading and writing. All data points of the same data point type to interact with each other are assigned to one group address.

    For the example, link "Group object 1" with "Rocker1 - Switching" to the group address "0/0/1 RockerSwitch1" and "Group object 2" with "LED left - Status indication" to "0/0/2 RockerLED".

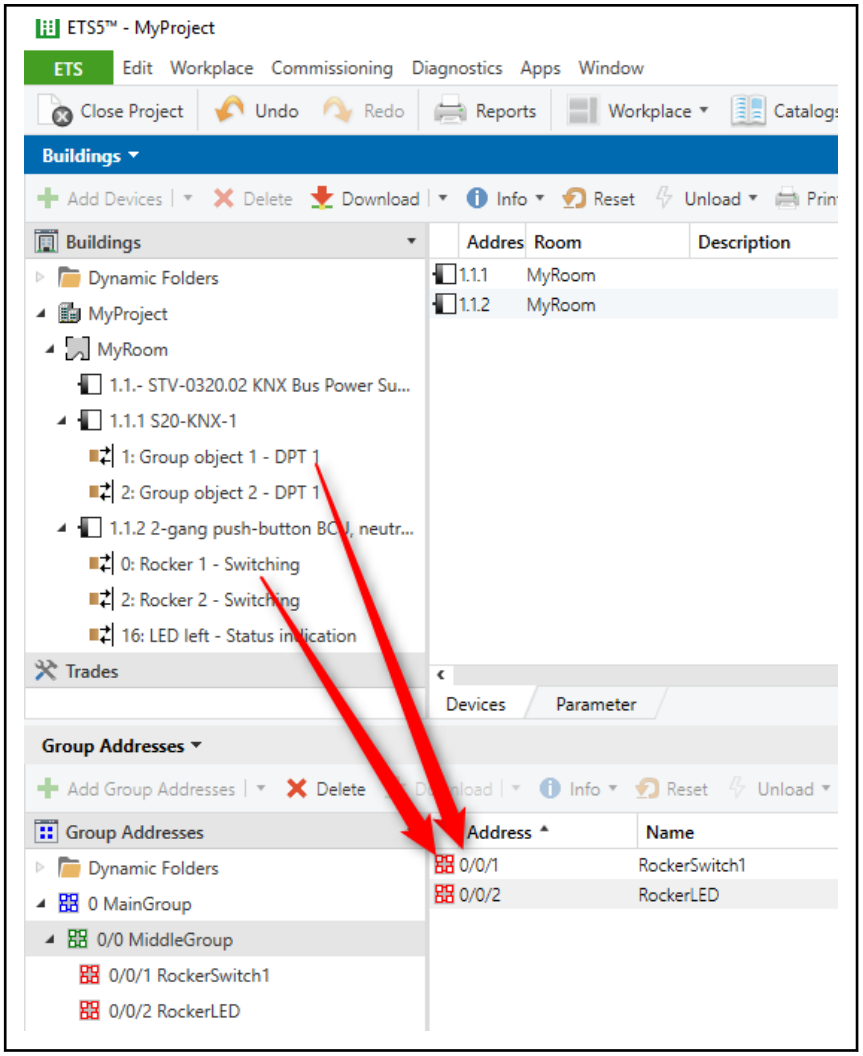


Fig. 24-14: Assigning data points

Without assigning data points to a group address, this data point cannot be used with the application S20-KNX-1 and causes the error "DPT is disabled in the KNX configuration (ETS)" when accessed.

The ETS project is now created and can be assigned to the KNX devices.

11. Upon the initial commissioning, the physical address has to be specified once for the KNX devices.
    1. Select **Download** ▸ **Download all**. Physical addresses are specified
    2. Follow the ETS instructions. During the download, press the programming pushbuttons of the KNX devices upon request

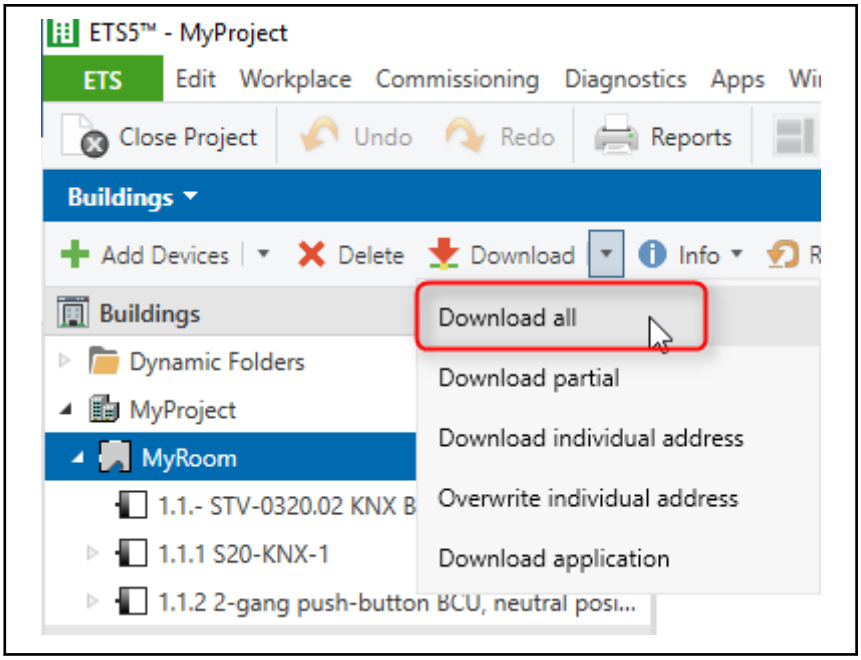


Fig. 24-15: Starting ETS download

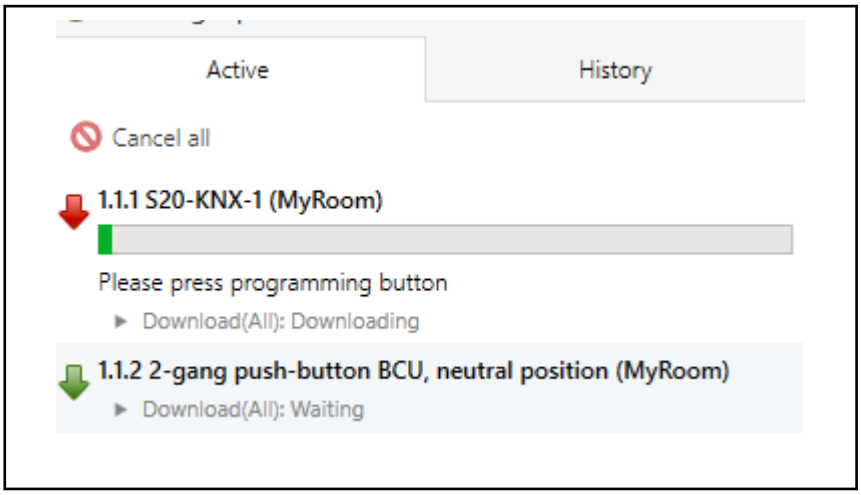


Fig. 24-16: Request to press the programming pushbutton on the KNX device (enable programming mode)

If the data points are again changed, a **Download ▸ Download partial** or a **Download ▸ Download application** is sufficient. The programming pushbutton has not to be pressed anymore.

☞ After a complete download or upon the initial commissioning, disconnect the module S20-KNX-1 from voltage. The control or the bus coupler with the module has to be disabled and enabled again.

If only one data point was added or changed, a restart is not required.

PLC application example

1. Add the **RIH_IndraControl_S20** library.
   Create the function block **IH_S20ComConfiguration**.

The function block provides the **S20BusHandle** to specify at which bus master (XM local or at the bus coupler) the S20-KNX-1 module is operated.

2. Add the **RIH_IndraControl_S20_KNX** library.

   Create the function block **IH_S20KNXClient**.

   The function blocks **IH_S20KNXClient** and **KNXHandle** provide the interface to the S20-KNX-1 module and the KNX data point type. A new function block instance of the **IH_S20KNXClient** is required for each S20-KNX-1 module.

3. To detect the pushbutton state, use the data point **DPT1** of the group "0/0/1 RockerSwtich1".

   Create the **IH_S20KNXDPT1** function block for each data point ID "DPTID" with the data point type DPT1 as own instance.

4. Example program code

   Reading the DPT ID 1 of the group 0/0/1 RockerSwitch1 to read the DPT ID 0 state that corresponds to the actual pushbutton position of the real pushbutton

   Writing an reading DPT ID 2 of the group 0/0/2 RockerLED to write and read the DPT ID 16 controlling the LED of the pushbutton

   Example ST

Program:

```
PROGRAM KNX_Example
VAR
    bInit                     : BOOL := TRUE;
    fbIH_S20ComConfiguration : IH_S20ComConfiguration; //FB Instance
    bEnableComConfig          : BOOL; //Enable
    enBusType                 : IH_S20_BUSTYPE := IH_S20_BUSTYPE.S20IO; //Bus type selection
    enBusConfig               : IH_S20_BUS_CONFIG; //Bus config for field bus device
    dwS20BusHandle            : DWORD;

    fbIH_S20KNXClient         : IH_S20KNXClient; //FB Instance
    bEnableClient             : BOOL; //Enable
    uiSlot: UINT              := 3; //Slot of the S20-KNX-1 module
    tTimeOut: TIME            := T#2S; //TimeOut
    stServerItems             : IH_S20_KNX_SERVERITEMS; //KNX Server Status
    enDPTStatus               : IH_S20_KNX_DPT_STATUS; //RAW DPT data
    enCommStatus              : IH_S20_KNX_STATUS; //State of the KNX Client FB
    dwKNXHandle               : DWORD; //KNX communication Handle for DPT FBs
    wStatusWord               : WORD; //Status of S20-KNX-1 module
    arInputComplexProtocol    : ARRAY [0..23] OF BYTE; //Input data of S20-KNX-1 module
    wControlWord              : WORD; //Control of S20-KNX-1 module
    arOutputComplexProtocol   : ARRAY [0..23] OF BYTE; //Output data of S20-KNX-1 module

    fbIH_S20KNXDPT1_R         : IH_S20KNXDPT1; //FB Instance
    bEnableDPT1_R             : BOOL;  //Enable
    bDone_DPT1_Read           : BOOL; //Done, indicates when the value is changed
    bValidDPT1_R              : BOOL; //Valid indicates the KNX valid flag of the DPTID
    bDPT1Value_R              : BOOL; //actual value of the KNX DPT

    fbIH_S20KNXDPT2_W         : IH_S20KNXDPT1; //FB Instance
    bEnableDPT1_W             : BOOL; //Enable
    bDone_DPT2_Write          : BOOL; //Done, indicates when the value is changed
    uiDPTID_DPT2: UINT        := 2; //DPT ID
    enReadWriteDPT            : IH_S20_KNX_RW := IH_S20_KNX_RW.WRITEAUTO;
                              //Write on change in SetValue and read after write success
    bSetValueDPT2             : BOOL; //Value which is written
    bValidDPT2_W              : BOOL; //Valid indicates the KNX valid flag of the DPTID
    bDPT2Value_R              : BOOL; //actual value of the KNX DPT
END_VAR
```

Program:

```
IF bInit THEN
    bEnableComConfig := TRUE;
```

```
    bEnableClient    := TRUE;
    bInit            := FALSE;
END_IF

fbIH_S20ComConfiguration(
    Enable         := bEnableComConfig,
    BusType        := enBusType,
    BusConfig      := enBusConfig,
    S20BusHandle   => dwS20BusHandle);
fbIH_S20KNXClient(
    Enable                := bEnableClient AND fbIH_S20ComConfiguration.InOperation,
                          //KNX Client can run, when ComConfiguration is running
    S20BusHandle          := dwS20BusHandle,
    Slot                  := uiSlot,
    StatusWord            := wStatusWord,
    InputComplexProtocol  := arInputComplexProtocol,
    TimeOut               := tTimeOut,
    ServerItems           => stServerItems,
    DPTStatus             => enDPTStatus,
    CommunicationStatus   => enCommStatus,
    KNXHandle             => dwKNXHandle,
    ControlWord           => wControlWord,
    OutputComplexProtocol => arOutputComplexProtocol);
//Read DPT ID 1 of Group 0/0/1 RockerSwitch1 to read DPT ID 0
fbIH_S20KNXDPT1_R(
    Enable     := bEnableDPT1_R AND fbIH_S20KNXClient.InOperation,
               //DPT FB can run when KNXClient is running
    Done       => bDone_DPT1_Read,
    DPTID      := 1,
    TimeOut    := tTimeOut,
    KNXHandle  := dwKNXHandle,
    Valid      => bValidDPT1_R,
    ActValue   => bDPT1Value_R);
//Write and Read DPT ID 2 of Group 0/0/2 RockerLED to write DPT ID 16
fbIH_S20KNXDPT2_W(
    Enable     := bEnableDPT1_W AND fbIH_S20KNXClient.InOperation,
               //DPT FB can run when KNXClient is running
    Done       => bDone_DPT2_Write,
    DPTID      := uiDPTID_DPT2,
    ReadWrite  := enReadWriteDPT,
    SetValue   := bSetValueDPT2,
    TimeOut    := tTimeOut,
    KNXHandle  := dwKNXHandle,
    Valid      => bValidDPT2_W,
    ActValue   => bDPT2Value_R);
```

Example CFC

Program:

```
PROGRAM KNX_Example_CFC
VAR
 fbIH_S20ComConfiguration   : IH_S20ComConfiguration; //FB Instance
 bEnableComConfig           : BOOL := TRUE;
 enBusType                  : IH_S20_BUSTYPE := IH_S20_BUSTYPE.S20IO; //Bus type selection
 enBusConfig                : IH_S20_BUS_CONFIG; //Bus config for field bus device

 fbIH_S20KNXClient          : IH_S20KNXClient; //FB Instance
 bEnableClient              : BOOL;
 enCommStatus               : IH_S20_KNX_STATUS; //State of the KNX Client FB
 wKNXStatus                 : WORD; //Status of S20-KNX-1 module
 wKNXControl                : WORD; //Control of S20-KNX-1 module
 arKNXComplexProtocolInput  : ARRAY [0..23] OF BYTE; //Input data of S20-KNX-1 module
 arKNXComplexProtocolOutput : ARRAY [0..23] OF BYTE; //Output data of S20-KNX-1 module

 fbIH_S20KNXDPT1_R          : IH_S20KNXDPT1; //FB Instance
 bEnableDPT1_R              : BOOL;

 fbIH_S20KNXDPT2_W          : IH_S20KNXDPT1; //FB Instance
 bEnableDPT1_W              : BOOL;
 enReadWriteDPT             : IH_S20_KNX_RW := IH_S20_KNX_RW.WRITEAUTO;
                              //Write on change in SetValue and read after write success
 bSetValueDPT2              : BOOL; //Value which is written
 uiDPTID_DPT2               : UINT := 2; //DPT ID
END_VAR
```

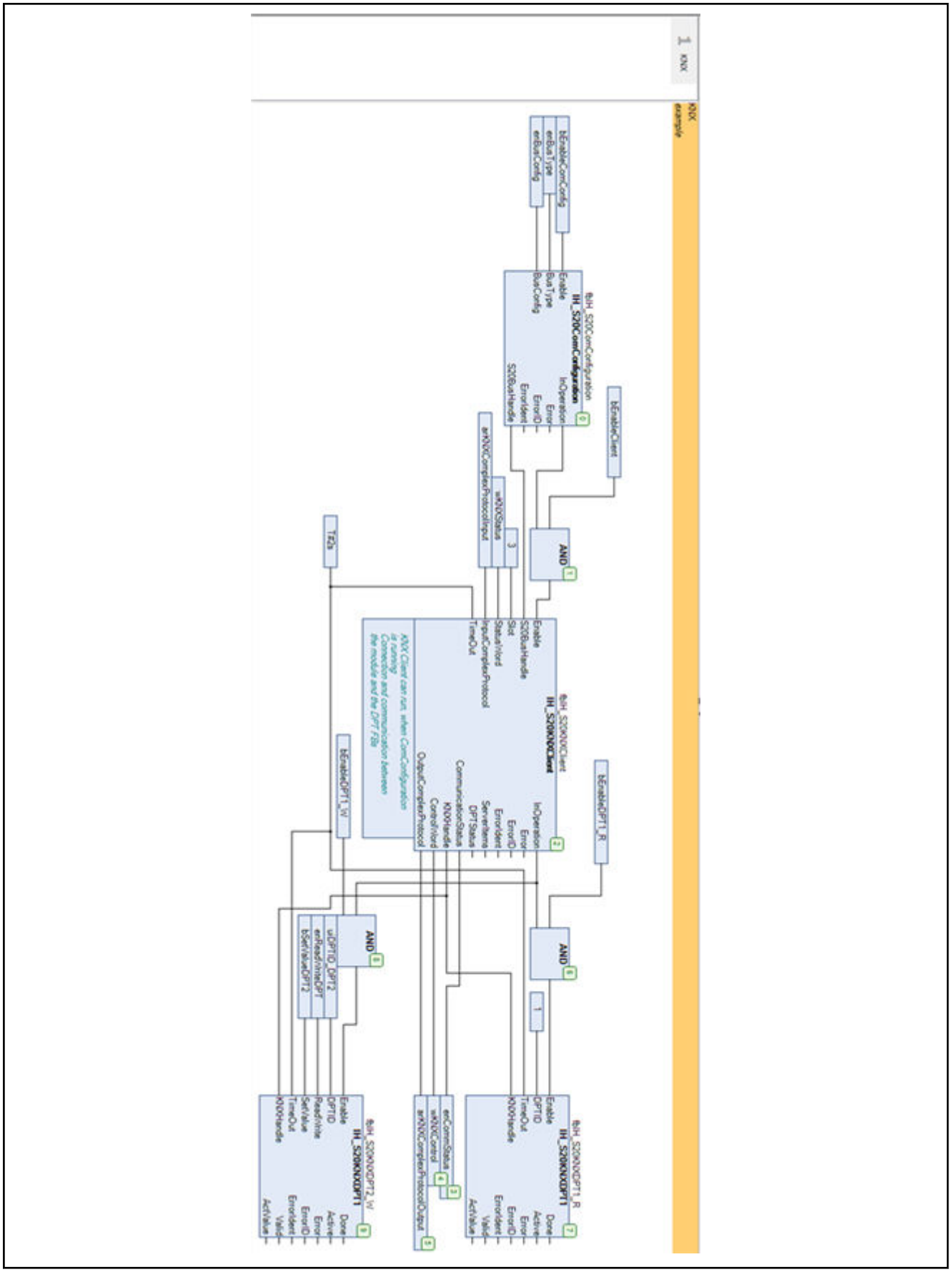


Fig. 24-17: Example CFC

25 BACnet libraries

25.1 Overview

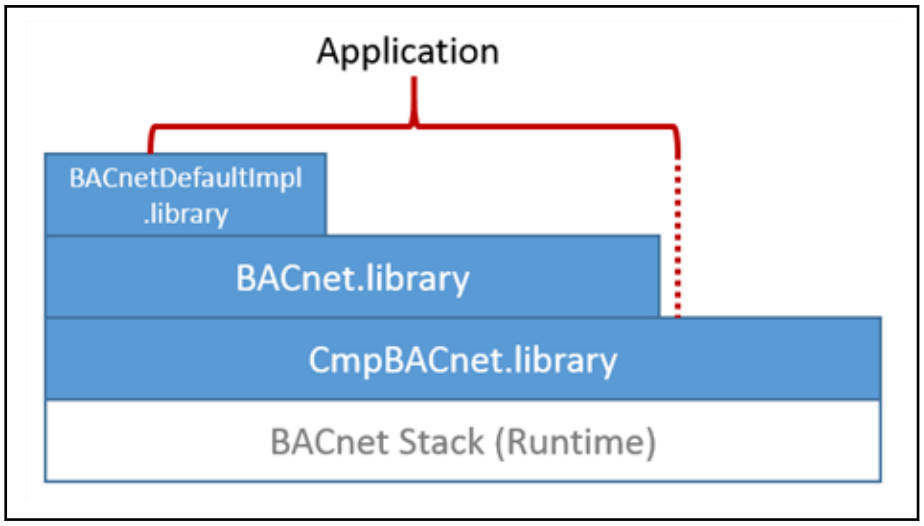


Fig. 25-1: Overview on BACnet libraries

BACnetDefaultImpl.library	Default implementations of use cases that can be used in the application if required, e.g. handler for the “Device Communication Control” or the “ReInit device”
BACnet.library	User interface for the supported server objects and client requests
CmpBACnet.library	BACnet stack integration (IEC interface). Only selected methods and special cases are useful for the application. Otherwise, use the function blocks of the BACnet.library

Tab. 25-1: Brief description of the BACnet libraries

Description of the BACnet libraries

For a detailed interface and functional description, go to the implemented help in the respective library or to the library manager.

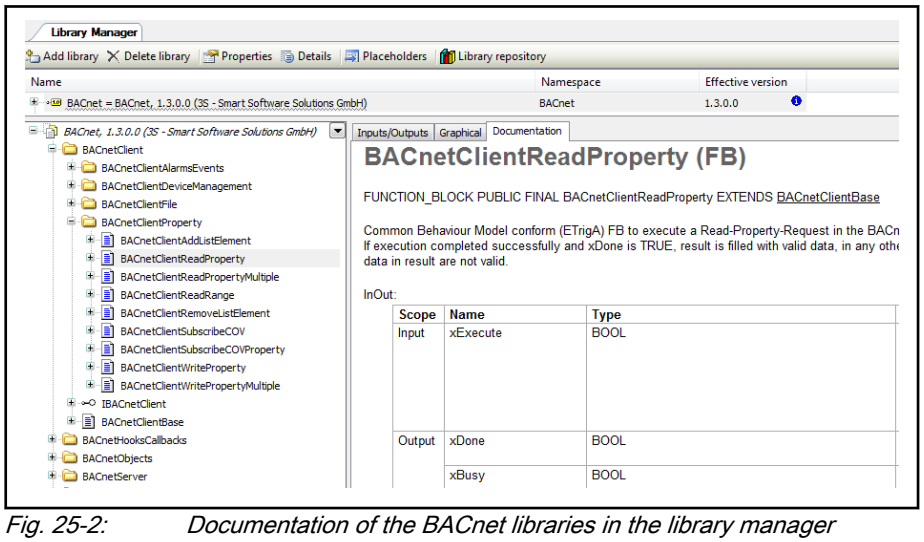


Fig. 25-2: Documentation of the BACnet libraries in the library manager

25.2 Programming examples

25.2.1 Server: NotificationClass → SetRecipientList

Program:

```
VAR
  rNotifyDest : BACnet.CmpBACnet.IEC_BACNET_DESTINATION;
END_VAR

rNotifyDest.recipient.tag                          := BACnet.CmpBACnet.IEC_BACNET_RECIPIENT_TYPE.RECIPIENT_DEVICE;
rNotifyDest.recipient.recipient.device.typ         := BACnet.CmpBACnet.IEC_BACNET_OBJECT_TYPE.OBJ_DEVICE;
rNotifyDest.recipient.recipient.device.instNumber := 1; // Instance-Number (Device-ID) from the Client
rNotifyDest.processID              := NC_1_ProcessId; // example
rNotifyDest.issueConfNotification := 1; // 0 = Unconfirmed / 1 = Confirmed
// Transitions (To-Offnormal, To-Fault, To-Normal)
BACnet.InitializeBACnetBitString( str:='111', bitStr:=rNotifyDest.transitions );
// Valid Week-Days
BACnet.InitializeBACnetBitString( str:='1111111', bitStr:=rNotifyDest.validDays );
rNotifyDest.fromTime.hour          := 0;
rNotifyDest.fromTime.minute        := 0;
rNotifyDest.fromTime.second        := 0;
rNotifyDest.fromTime.hundredths    := 0;
rNotifyDest.toTime.hour            := 23;
rNotifyDest.toTime.minute          := 59;
rNotifyDest.toTime.second          := 59;
rNotifyDest.toTime.hundredths      := 99;

NC_1.SetRecipientList( 1, ADR(rNotifyDest) );
```

25.2.2 Server/client: IBACnetEventConsumer → BACnetEventCallback

Use the IBACnetEventConsumer interface (BACnetEventCallback method) can be used to event-control different events.

The following example code registers COV notifications. It applies exemplarily to all events (see CmpBACnet.library → IEC_BACNET_CB_TYPE in the BACnetEnums folder):

- Create a function block implementing the IBACnetEventConsumer interface
- The outputs and variables in the following example code are exemplarily shown for event displayed

Program:

```
FUNCTION_BLOCK FB_CB_CONF_COV_NOTIFICATION IMPLEMENTS BACnet.IBACnetEventConsumer
VAR_OUTPUT
  EvCnt               : UINT := 0; // call count
  LastEvtValid        : BOOL := FALSE;
  LastCovNotifyInfo   : BACnet.CmpBACnet.IEC_BACNET_COV_NOTIF_INFO;
  LastPropertyValue   : BACnet.CmpBACnet.IEC_BACNET_PROPERTY_VALUE;
  LastRealValue       : REAL;
END_VAR
VAR
  prEvtConfCovNotify  : POINTER TO BACnet.CmpBACnet.EVT_BACNET_CONFCOVNOTIFICATION;
END_VAR
```

- This interface specifies the BACnetEventCallback method provided with generic transfer parameters
- To process the respective parameters of the specific event, the generic pointer “pParameter” has to be casted to the respective event structure
- In this example, event information is then copied to the function block

Program:

```
METHOD BACnetEventCallback : BOOL
VAR_INPUT
```

```
  EventId   : DWORD; // ID of the occured event (see CmpBACnet.library -> GVL EVENTIDS)
  usParamId : WORD;  // ID of the parameter structure (see CmpBACnet.library -> GVL EVENTIDS)
  usVersion : WORD;  // Version of hte parameter structure (see CmpBACnet.library -> GVL EVENTIDS)
  pParameter: POINTER TO BYTE; // Pointer to the event specific parameter, that is specified by ID
                               // (e.g. EVT_BACNET_ACKALARM)
END_VAR

// NOTE: no breakpoint possible, because this event callback is called from the BACstack task,
// which is not an IEC task!

// Check EventId
IF (    EventId   = BACnet.CmpBACnet.EVT_ConfCOVNotification
    AND usParamId = BACnet.CmpBACnet.EVTPARAMID_CmpBACnet_ConfCOVNotification
) THEN
  LastEvtValid := TRUE;
  EvCnt := EvCnt + 1;

  // Event parameter
  prEvtConfCovNotify   := pParameter;
  LastCovNotifyInfo    := prEvtConfCovNotify^.pServiceInfo^;
  LastPropertyValue    := LastCovNotifyInfo.listOfValues^;
  // Note: the buffer in 'PropertyContents' is stack internal, that means you have to copy
  // the value here in the callback!
  LastRealValue := BACnet.GetRealFromContents(LastPropertyValue.value);

ELSE // unexpected EventId
  LastEvtValid := FALSE;

END_IF


// Return
BACnetEventCallback := TRUE; // TRUE = no calls of other registered callbacks of this event

(* Note
prEvtConfCovNotify^.pCBStatus^ := BACnet.CmpBACnet.IEC_BACNET_CB_STATUS.CB_STATUS_DEFAULT
  => default: the stack handles the default actions (and not the hook / callback)

prEvtConfCovNotify^.pCBStatus^ := BACnet.CmpBACnet.IEC_BACNET_CB_STATUS.CB_STATUS_OK
  => the hook or application generates the reply, see the ((Response-)-'CbCompletion' functions,
  which are mentioned in the corresponding event strukturen,
  e.g. see documentation of EVT_BACNET_CONFCOVNOTIFICATION.
*)
```

- Create an instance of this function block in a program function block:

  `fbCbConfCovNotification : FB_CB_CONF_COV_NOTIFICATION;`

- This function block instance is registered for a specific event type (→ IEC_BACNET_CB_TYPE) at the BACnet device and can also be unregistered again (UnregisterHook):

  `eBACnetStatus := BACnet_Server.RegisterHook( BACnet.CmpBACnet.IEC_BACNET_CB_TYPE.CB_CONF_COV_NOTIFICATION, fbCbConfCovNotification );`



For more examples, refer to the general BACnet description in the documentation “IndraWorks 15VRS Field Buses” (DOK-IWORKS-FB******V15-APRS-EN-P).

26 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

Abbreviations 15
About this documentation 1
  Validity of the documentation 1
Active 18
ANSI Z535.6-2006 4
Application phases 2

B

BACnet libraries
  BACnet.library 967
  BACnetDefaultImpl.library 967
  CmpBACnet.library 967
  Overview 967
  Programming examples 968

C

Change record 1
CommandAborted 18
CommunicationState 211
Complaints 16
ConfigState 210
Configuration
  R IB-IL-AO-2/SF 610
  R IB-IL-AO-2/U/BP 610
  R-IB IL AI 2/SF 560
  R-IB IL AI 4/EF 564
  R-IB IL AI 4/I 569
  R-IB IL AI 4/U 569
  R-IB IL AI8/IS 573
  R-IB IL AI8/SF 573
  R-IB IL AO 2/UI-PAC 614
  R-IB IL CNT 618, 624, 630, 634
  R-IB IL INC-IN-PAC 672
  R-IB IL PWM/2-PAC 639, 642, 646, 649
  R-IB IL RS232 PRO 677
  R-IB IL RS485/422 PRO 677
  R-IB IL SGI 2/F-PAC 578
  R-IB IL SSI 652
  R-IB IL SSI IN-PAC 667
  R-IB IL TEMP 2 UTH-PAC 584
  R-IB IL TEMP 4/8 RTD-EF-PAC 601
  R-IB IL TEMP 4/8 RTD-PAC 595
  R-IB IL TEMP2 RTD-PAC 589
  R-IL_AO1Sf 607
Criticism 16
Customer Feedback 16

D

Documentation
  Revision history 1
Done 18
DP_DEVICELIST 36
DP_MASTERINFO 37
DP_MASTERLIST 36
DP_SLAVELIST 35

E

Enable 18
Error codes
  IH_PBDPAck_PSDx_Inline 700
  IH_PBDPAck_PSDx_S20 705
  IH_PNIOAck_PSDx_Inline 717
  IH_PNIOAck_PSDx_S20 712
  IH_SIIIAck_SSDx_S20 724
  IL_AI2Sf 564
  IL_AI4Ef 569
  IL_AI4IU 573
  IL_AI8SfIs 578
  IL_AO1Sf 610
  IL_AO2SfBp 614
  IL_AO2UI 618
  IL_CntCount 624
  IL_CntFrequency 630
  IL_CntPulse 633
  IL_CntTime 639
  IL_DPMasterState 39
  IL_ENIPAdapterBackup 229
  IL_ENIPAdapterRestore 232
  IL_ENIPAdapterState 247
  IL_ENIPAdapterStateDetails 248
  IL_IncIn 676
  IL_PBDPSlaveDPV1GetWriteData 66
  IL_PBDPSlaveDPV1Polling 61
  IL_PBDPSlaveDPV1Response 64
  IL_PNIOAssignName 91
  IL_PNIOCheckTopology 96
  IL_PNIOControllerState 116
  IL_PNIOControllerStateDetails 117
  IL_PNIODeviceBackup 85, 86
  IL_PNIODeviceReplacement 98
  IL_PNIODeviceResetToDefault 100
  IL_PNIODeviceRestore 88, 89
  IL_PNIODeviceSetIP 102
  IL_PNIODeviceState 207
  IL_PNIODeviceStateDetails 208
  IL_PNIODeviceStateDetailsXMAC 209
  IL_PNIODiagnosis 122
  IL_PNIOGetConfigDeviceNameList 124
  IL_PNIOGetControllerList 125
  IL_PNIOGetDiagDeviceNameList 126
  IL_PNIOGetLiveList 104
  IL_PNIOGetLogMsg 105
  IL_PNIOGetPortData 107, 108
  IL_PNIOIoLinkBackupDevice 180
  IL_PNIOIoLinkCall 144
  IL_PNIOIoLinkGetDeviceAccessLock 149
  IL_PNIOIoLinkIdent 153
  IL_PNIOIoLinkIdentDetails 159
  IL_PNIOIoLinkRemoteDeviceState 164

IL_PNIOIoLinkRestoreDevice..... 183
IL_PNIOIoLinkSetDeviceAccessLock..... 169
IL_PNIOIoLinkSystemCommand..... 174
IL_PNIOReadRecord..... 73
IL_PNIOReadWriteIMData..... 136
IL_PNIORemoteDeviceState..... 128
IL_PNIORemoteDeviceStateDetails..... 130
IL_PNIORestoreTopologie..... 114
IL_PNIOServiceRecord..... 78
IL_PNIOStoreTopologie..... 109
IL_PNIOWriteRecord..... 81
IL_Pwm2Frequency..... 642
IL_Pwm2PulseDirection..... 649
IL_Pwm2Pwm..... 645
IL_Pwm2SinglePulse..... 652
IL_RsCom..... 684
IL_Sgi2f..... 584
IL_SIIICommand..... 347
IL_SIIIGetConfigSlaveAdrList..... 376
IL_SIIIGetDiagSlaveAdrList..... 377
IL_SIIIGetLiveSlaveAdrList..... 379
IL_SIIIGetSercosPhase..... 410
IL_SIIIMasterState..... 380
IL_SIIIMasterStateDetails..... 381
IL_SIIIParameterBackup..... 367
IL_SIIIParameterRestore..... 373
IL_SIIIReadParameter..... 350
IL_SIIIReadRealParameter..... 354
IL_SIIIReadStringParameter..... 356
IL_SIIIRemoteSlaveState..... 383
IL_SIIIRemoteSlaveStateDetails..... 385
IL_SIIISetSercosPhase..... 412
IL_SIIISvcRead..... 335
IL_SIIISvcWrite..... 340
IL_SIIIWriteParameter..... 358
IL_SIIIWriteRealParameter..... 361
IL_SlaveMapListAddEntry..... 327
IL_SlaveMapListInit..... 324
IL_Ssi..... 667
IL_SsiIn..... 672
IL_Status..... 249
IL_Temp2Rtd..... 595
IL_Temp2Uth..... 589
IL_Temp4_8Rtd..... 601
IL_Temp4_8RtdEf..... 607
MB_SIIIDRn5020..... 541
MB_SIIIFcP5020..... 545
MB_SIIIPlcDriveDirectValveControl..... 505
MB_SIIIPlcDriveEfcFrequencyType01..... 481
MB_SIIIPlcDriveFlowControl..... 487
MB_SIIIPlcDrivePosInterpolationType01..... 444
MB_SIIIPlcDrivePositioningType01..... 439
MB_SIIIPlcDrivePressureDirectValve-Control..... 521
MB_SIIIPlcDrivePressureFlowControl..... 492
MB_SIIIPlcDrivePressureSwivelAngle-Slave..... 501
MB_SIIIPlcDriveTorqueDirectValveCon-trol..... 511
MB_SIIIPlcDriveTorqueFlowControl..... 516
MB_SIIIPlcDriveTorqueType01..... 434
MB_SIIIPlcDriveType01..... 426
MB_SIIIPlcDriveVelocityAltCtrl..... 526
MB_SIIIPlcDriveVelocityType01..... 430
MB_SIIIPlcPressureSwivelAngleControl..... 497
MB_SIIIPowerSupplyDcDcIslandGrid..... 469
MB_SIIIPowerSupplyDriveIntegrated..... 455
MB_SIIIPowerSupplyStandalone 463
Profinet I/O controller..... 193
RIH_Dali..... 784
RIH_Modbus..... 869
RIL_EtherNetIP..... 243
Error tables
ETHERCAT_TABLE..... 318
ETHERNET_IP_TABLE..... 242, 251
PB_DP_TABLE Profibus DP error..... 52
PROFINETIO_TABLE..... 190, 213
Error_Code_DP..... 55
Error_Code_FDL..... 54
Error_Code_OB..... 54
Error_Tables
ETHERCAT_TABLE..... 318
ETHERNET_IP_TABLE..... 242, 251
PB_DP_TABLE Profibus DP error..... 52
PROFINETIO_TABLE..... 190, 213
ErrorID
RIL_ProfibusDP_02..... 51
ErrorIdent..... 52
ETHERCAT_TABLE..... 318
EtherNet/IP adapter
Advanced error codes for Connection Manager..... 255
CIP, General error codes..... 253
Error codes, general..... 251
Error codes, special..... 253
Execute..... 18
ExecuteLock..... 18

F

Feedback..... 16
FM_Error_Code..... 53
Function block types..... 17
Function block types and their interfaces..... 18

G

General error codes
Profinet I/O controller..... 191
PROFINET I/O device..... 213

H

Hazard warnings..... 3
Helpdesk..... 971
Hotline..... 971

I

IH_DaliAddress ... 728
IH_DaliAddToGroup ... 747
IH_DaliAddToScene ... 749
IH_DaliCallScene ... 730
IH_DaliDim ... 726
IH_DaliDirectPower ... 734
IH_DaliOff ... 751
IH_DaliOn ... 753
IH_DaliProgramShortAddress ... 759
IH_DaliQueryActualLevel ... 744
IH_DaliQueryDTR ... 769
IH_DaliQueryGroup ... 774
IH_DaliQueryParameter ... 771
IH_DaliQueryRandomAddress ... 777
IH_DaliQuerySceneLevel ... 757
IH_DaliQueryStatus ... 738
IH_DaliReadWrite ... 742
IH_DaliRemoveFromGroup ... 732
IH_DaliRemoveFromScene ... 736
IH_DaliReset ... 766
IH_DaliSearchAddress ... 755
IH_DaliStoreParameter ... 763
IH_INLN_DALI_ADDRTYPE ... 781
IH_INLN_DALI_DIMMODE ... 781
IH_INLN_DALI_LIGHTLEVEL ... 781
IH_INLN_DALI_OPMODE ... 782
IH_INLN_DALI_PARAMETER ... 783
IH_INLN_DALI_TOKEN ... 780
IH_INLN_MB_CONFIG ... 865
IH_INLN_MB_DATA ... 864
IH_INLN_MB_DEV_IDCODE ... 864
IH_INLN_MB_EVENTLOG ... 867
IH_INLN_MB_FIFODATA ... 865
IH_INLN_MB_RECORD ... 866
IH_INLN_MB_SUBREQUESTS ... 866
IH_INLN_MB_TOKEN ... 866
IH_MBGetCommEventCounter (FC 11) ... 787
IH_MBGetCommEventLog (FC 12) ... 790
IH_MBMaskWriteRegister (FC 22) ... 794
IH_MBReadCoils (FC 1) ... 797
IH_MBReadDeviceIdentification (FC 43) ... 801
IH_MBReadDiagnostics (FC 8) ... 807
IH_MBReadDiscreteInputs (FC 2) ... 810
IH_MBReadExceptionStatus (FC 7) ... 814
IH_MBReadFIFOQue (FC 24) ... 816
IH_MBReadFileRecord (FC 20) ... 819
IH_MBReadHoldingRegisters (FC 3) ... 823
IH_MBReadInputRegisters (FC 4) ... 827
IH_MBReadWriteMultipleRegisters (FC 23) ... 831
IH_MBRTUMaster ... 835
IH_MBRTUSlave ... 840
IH_MBWriteFileRecord (FC 21) ... 846
IH_MBWriteMultipleCoils (FC 15) ... 851
IH_MBWriteMultipleRegisters (FC 16) ... 854
IH_MBWriteSingleCoil (FC 5) ... 858
IH_MBWriteSingleRegister (FC 6) ... 861
IH_PBDPAck_PSDx_Inline ... 695
IH_PBDPAck_PSDx_S20 ... 700
IH_PNIOAck_PSDx_Inline ... 713
IH_PNIOAck_PSDx_S20 ... 707
IH_S20AI4IU ... 912
IH_S20AI4RTD ... 918
IH_S20AI4UTH ... 920
IH_S20AI8 ... 916
IH_S20AI8RTD ... 922
IH_S20AI8UTH ... 924
IH_S20AO4 ... 928
IH_S20AO8 ... 926
IH_S20CntInc ... 935
IH_S20ComConfiguration ... 904
IH_S20ComInterface ... 906
IH_S20ComInterfaceType02 ... 905
IH_S20IOModuleStateDetails ... 900
IH_S20IOState ... 901
IH_S20IOStateDetails ... 902
IH_S20ModuleIdentDetails ... 903
IH_S20RSCom ... 940
IH_S20SsiAo1 ... 931
IH_SIIIAck_SSDx_S20 ... 719
IL_ADDRESSTYPE ... 327
IL_AI2Sf ... 560
IL_AI4Ef ... 564
IL_AI4IU ... 569
IL_AI8SfIs ... 573
IL_AO1Sf ... 607
IL_AO2SfBp ... 610
IL_AO2UI ... 614
IL_BACKUP_ERRORS ... 374
IL_BACKUP_OPTIONS ... 374
IL_BUSMASTER ... 31, 184, 239, 303, 415
IL_BUSSLAVE ... 66, 212, 250, 329
IL_CntCount ... 618
IL_CntFrequency ... 624
IL_CntPulse ... 630
IL_CntTime ... 634
IL_DPBaudrateGet ... 48
IL_DPDeviceListGet ... 47
IL_DPDevInfoGet ... 48
IL_DPIdent ... 39
IL_DPMasterState ... 37
IL_DPPrjSlaveListGet ... 49
IL_DPReadDiag ... 43
IL_DPSlaveDiagListGet ... 50
IL_DPSycFr ... 45
IL_DPV1Read ... 40
IL_DPV1Write ... 41
IL_ECAT_ADDRESS_TYPE ... 303
IL_ECAT_BASE_DATA_TYPE ... 303
IL_ECAT_COE_DIAG_HISTORY_FLAGS ... 305
IL_ECAT_COE_DIAG_HISTORY_STRUCT ... 305
IL_ECAT_COE_DIAG_MSG_EXT ... 306
IL_ECAT_COE_DIAG_MSG_RAW ... 307
IL_ECAT_COE_DIAG_TYPE ... 308
IL_ECAT_DL_STATUS ... 308

IL_ECAT_ESC_FEATURES..... 308
IL_ECAT_MASTER_BUS_STATISTIC..... 309
IL_ECAT_MASTER_STATE..... 309
IL_ECAT_MBX_CONFIG_PROTOCOLS..... 310
IL_ECAT_MBX_ONLINE_PROTOCOLS..... 311
IL_ECAT_MBX_SIZE_INFO..... 311
IL_ECAT_MEMORY_INFO..... 311
IL_ECAT_PORT_ERROR_COUNTER..... 311
IL_ECAT_PORT_STATE..... 312
IL_ECAT_SLAVE_CONFIG_INFO..... 313
IL_ECAT_SLAVE_ERROR_COUNTER..... 314
IL_ECAT_SLAVE_IDENTITY_INFO..... 314
IL_ECAT_SLAVE_ONLINE_INFO..... 314
IL_ECAT_SLAVE_STATE..... 315
IL_ECAT_SOE_ATTRIBUTE..... 316
IL_ECAT_SOE_DATA_LENGTH..... 316
IL_ECAT_SOE_DATA_STATUS..... 317
IL_ECAT_SOE_DATA_TYPE..... 317
IL_ECAT_SOE_ELEMENT..... 317
IL_ECAT_SOE_PARAM_TYPE..... 318
IL_ECAT_STATE..... 318
IL_ECATCoeReadDiagHistory..... 290
IL_ECATCoeReadSdo..... 258
IL_ECATCoeWriteSdo..... 261
IL_ECATGetConfigEthercatAddrList..... 273
IL_ECATGetNumConfigSlaves..... 271
IL_ECATGetNumOnlineSlaves..... 272
IL_ECATGetOnlineEthercatAddrList..... 275
IL_ECATGetSlaveConfigInfo..... 276
IL_ECATGetSlaveOnlineInfo..... 278
IL_ECATMasterBusStatistic..... 281
IL_ECATMasterGetBusState..... 295
IL_ECATMasterSetBusState..... 296
IL_ECATMasterState..... 280
IL_ECATMasterStateDecoding..... 302
IL_ECATRemoteSlaveClearErrorCounter..... 289
IL_ECATRemoteSlaveGetDevState..... 298
IL_ECATRemoteSlaveGetErrorCounter..... 288
IL_ECATRemoteSlaveGetErrorCounterPeriod..... 286
IL_ECATRemoteSlaveSetDevState..... 299
IL_ECATRemoteSlaveSetErrorCounterPeriod..... 285
IL_ECATRemoteSlaveState..... 283
IL_ECATSlaveStateDecoding..... 302
IL_ECATSoeAttributeDecoding..... 301
IL_ECATSoeCommand..... 269
IL_ECATSoeIdnCoding..... 301
IL_ECATSoeRead..... 263
IL_ECATSoeWrite..... 267
IL_ENIP_ADAPTER_STATE..... 250
IL_ENIP_IDENTITY_ATTRIBUTES..... 239
IL_ENIP_REMOTE_ADAPTER_STATE_DETAILS..... 240
IL_ENIP_SCANNER_STATE..... 241
IL_ENIPAdapterBackup..... 226
IL_ENIPAdapterRestore..... 230
IL_ENIPAdapterState..... 245
IL_ENIPAdapterStateDetails..... 247
IL_ENIPGetAttributeSingle..... 218
IL_ENIPGetIdentityObjAllAttributes..... 220
IL_ENIPRemoteAdapterState..... 232
IL_ENIPRemoteAdapterStateDetails..... 233
IL_ENIPScannerState..... 236
IL_ENIPScannerStateDetails..... 237
IL_ENIPSendUnconnectedMessage..... 222
IL_ENIPSetAttributeSingle..... 224
IL_FBUS_COMMUNICATION_STATE.... 185, 242, 405
IL_FBUS_SLAVE_RESULT..... 67
IL_FBUS_SLAVE_SERVICE..... 67
IL_FBUS_SLAVE_STATE..... 185, 242
IL_FIELDBUSOBJECT..... 327
IL_FIELDBUSTYPE..... 328
IL_IncIn..... 672
IL_INLN_AI_CONFIG
Declaration..... 684
IL_AI2Sf..... 561
IL_AI4Ef..... 566
IL_AI4IU..... 570
IL_AI8SfIs..... 575
IL_INLN_AI_DATA..... 685
IL_AI2Sf..... 562
IL_AI4Ef..... 566
IL_AI4IU..... 571
IL_AI8SfIs..... 575
IL_INLN_AI_RANGE
Declaration..... 684
IL_AI2Sf..... 562
IL_AI4Ef..... 567
IL_AI4IU..... 572
IL_AI8SfIs..... 576
IL_INLN_AO_CONFIG
IL_AO2UI..... 615
IL_INLN_AO_DATA
IL_AO2SfBp..... 611
IL_INLN_AO_RANGE
IL_AO2UI..... 616
IL_INLN_CNT_COUNT
IL_CntCount..... 620
IL_INLN_CNT_FREQ
IL_CntFrequency..... 626
IL_INLN_CNT_FREQ_CONTROL
IL_CntFrequency..... 626
IL_INLN_CNT_GATE
IL_CntCount..... 622
IL_INLN_CNT_INCONFIG
Declaration..... 686
IL_CntCount..... 621
IL_CntFrequency..... 627
IL_CntTime..... 636
IL_INLN_CNT_OUTBEHAVE
IL_CntCount..... 622
IL_INLN_CNT_PULSE_FREQ
IL_CntPulse..... 632
IL_INLN_CNT_PULSE_WIDTH

IL_CntCount ... 622
IL_INLN_CNT_SG_OPERATION
IL_CntCount ... 621
IL_CntFrequency ... 627
IL_CntTime ... 636
IL_INLN_CNT_TIME
IL_CntTime ... 635
IL_INLN_CNT_TIME_OUT_RELATION
IL_CntTime ... 637
IL_INLN_CNT_TIME_RES
IL_CntTime ... 637
IL_INLN_DIAG_CODE
Declaration ... 685
IL_AI2Sf ... 562
IL_AI4Ef ... 567
IL_AI4IU ... 571
IL_AI8SfIs ... 576
IL_AO1Sf ... 608
IL_AO2SfBp ... 612
IL_AO2UI ... 616
IL_Sgi2f ... 581
IL_Temp2Rtd ... 592
IL_Temp2Uth ... 587
IL_Temp4_8Rtd ... 598
IL_Temp4_8RtdEf ... 604
IL_INLN_INC_CONFIG
IL_IncIn ... 673
IL_INLN_INC_DATA
IL_IncIn ... 674
IL_INLN_INC_MODE
IL_IncIn ... 674
IL_INLN_PWM_PULSE
IL_Pwm2Pwm ... 644
IL_INLN_PWM_TIME_BASE
IL_Pwm2SinglePulse ... 651
IL_INLN_RS_BAUDRATE ... 690
IL_INLN_RS_CONFIG ... 689
IL_INLN_RS_DATAWIDTH ... 691
IL_INLN_RS_DIAG ... 692
IL_INLN_RS_INTERFACE ... 691
IL_INLN_RS_MODULETYPE ... 692
IL_INLN_RS_PROTOCOL ... 690
IL_INLN_SF_BP
IL_AO2SfBp ... 612
IL_INLN_SGI_CONFIG
IL_Sgi2f ... 580
IL_INLN_SGI_DATA
IL_Sgi2f ... 581
IL_INLN_SGI_RANGE
IL_Sgi2f ... 581
IL_INLN_SSI_CONFIG
IL_Ssi ... 655
IL_INLN_SSI_CTRL
IL_Ssi ... 658
IL_INLN_SSI_DATA
IL_Ssi ... 660
IL_INLN_SSI_FREQ
IL_SsiIn ... 670
IL_INLN_SSI_OPMODE
IL_Ssi ... 660
IL_INLN_SSI_PARITY
IL_SsiIn ... 669
IL_INLN_SSI_PATH
IL_Ssi ... 657
IL_INLN_TEMP_CONFIG
Declaration ... 686
IL_Temp2Rtd ... 590
IL_Temp4_8Rtd ... 596
IL_INLN_TEMP_CONNECTIONTYPE
Declaration ... 686
IL_Temp2Rtd ... 593
IL_Temp4_8Rtd ... 599
IL_INLN_TEMP_DATA
IL_Temp2Rtd ... 591
IL_Temp2Uth ... 586
IL_Temp4_8Rtd ... 598
IL_Temp4_8RtdEf ... 604
IL_INLN_TEMP_EF_CONFIG
Declaration ... 688
IL_Temp4_8RtdEf ... 603
IL_INLN_TEMP_EF_ODS
Declaration ... 688
IL_Temp4_8RtdEf ... 605
IL_INLN_TEMP_EF_R0
Declaration ... 688
IL_Temp4_8RtdEf ... 605
IL_INLN_TEMP_EF_SENSOR
Declaration ... 689
IL_Temp4_8RtdEf ... 605
IL_INLN_TEMP_R0
Declaration ... 687
IL_Temp2Rtd ... 592
IL_Temp4_8Rtd ... 598
IL_INLN_TEMP_SENSOR
Declaration ... 687
IL_Temp2Rtd ... 593
IL_Temp4_8Rtd ... 599
IL_INLN_UTH_CJCOMP
IL_Temp2Uth ... 587
IL_INLN_UTH_CONFIG
IL_Temp2Uth ... 586
IL_INLN_UTH_SENSOR
IL_Temp2Uth ... 587
IL_IOL_BACKUP_MODE ... 139
IL_IOL_DEVICE_STATUS ... 137
IL_IOL_DEVICE_STATUS_DETAILS ... 137
IL_IOL_EVENT_QUALIFIER ... 137
IL_IOL_EVENT_QUALIFIER_INSTANCE ... 138
IL_IOL_EVENT_QUALIFIER_MODE ... 138
IL_IOL_EVENT_QUALIFIER_SOURCE ... 138
IL_IOL_EVENT_QUALIFIER_TYPE ... 138
IL_IOL_IN_PROC_SPOOL_TYPE01L ... 886
IL_IOL_OUT_PROC_SPOOL_TYPE01 ... 885
IL_IOL_RESTORE_MODE ... 139
IL_IOL_SERVICE ... 139
IL_IoLinkCall ... 875

IL_IoLinkDirectValveControlType01..... 882
IL_IoLinkErrorText..... 880
IL_IoLinkIdent..... 880
IL_PBDPSlaveDPV1GetWriteData..... 64
IL_PBDPSlaveDPV1Polling..... 59
IL_PBDPSlaveDPV1Response..... 61
IL_PNIO_CONTROLLER_STATE..... 185
IL_PNIO_DEVICE_STATE..... 209
IL_PNIO_DEVICE_XMAC..... 211
IL_PNIO_DEVICE_XMAC_PORT..... 211
IL_PNIO_IM_RWMODE..... 131
IL_PNIO_IM0..... 130
IL_PNIO_IM1..... 131
IL_PNIO_IM2..... 131
IL_PNIO_IM3..... 131
IL_PNIO_IM4..... 131
IL_PNIO_PORT_DATA..... 186
IL_PNIO_REMOTE_DEVICE_INFO..... 186
IL_PNIO_REMOTE_DEVICE_STATE..... 187
IL_PNIO_TOPO_STATE..... 187
IL_PNIO_TOPO_STATUS..... 188
IL_PNIOAssignName..... 89
IL_PNIOCheckTopology..... 91
IL_PNIOControllerState..... 114
IL_PNIOControllerStateDetails..... 116
IL_PNIODeviceBackup..... 82
IL_PNIODeviceReplacement..... 96
IL_PNIODeviceResetToDefault..... 98
IL_PNIODeviceRestore..... 86
IL_PNIODeviceSetIP..... 100
IL_PNIODeviceState..... 205
IL_PNIODeviceStateDetails..... 207
IL_PNIODeviceStateDetailsXMAC..... 208
IL_PNIODiagnosis..... 118
IL_PNIOGetConfigDeviceNameList..... 122
IL_PNIOGetControllerList..... 124
IL_PNIOGetDiagDeviceNameList..... 125
IL_PNIOGetLiveList..... 102
IL_PNIOGetLogMsg..... 104
IL_PNIOGetPortData..... 105
IL_PNIOIoLinkBackupDevice..... 175
IL_PNIOIoLinkCall..... 139
IL_PNIOIoLinkGetDeviceAccessLock..... 145
IL_PNIOIoLinkIdent..... 150
IL_PNIOIoLinkIdentDetails..... 155
IL_PNIOIoLinkRemoteDeviceState..... 160
IL_PNIOIoLinkRestoreDevice..... 180
IL_PNIOIoLinkSetDeviceAccessLock..... 165
IL_PNIOIoLinkSystemCommand..... 170
IL_PNIOReadRecord..... 71
IL_PNIOReadWriteIMData..... 132
IL_PNIORemoteDeviceState..... 126
IL_PNIORemoteDeviceStateDetails..... 128
IL_PNIORestoreTopology..... 109
IL_PNIOServiceRecord..... 74
IL_PNIOStoreTopology..... 108
IL_PNIOWriteRecord..... 79
IL_PSDx_ACK (old)..... 695
IL_Pwm2Frequency..... 639
IL_Pwm2PulseDirection..... 646
IL_Pwm2Pwm..... 642
IL_Pwm2SinglePulse..... 649
IL_RsCom..... 677
IL_SERCOS_DATA_STATUS..... 361
IL_Sgi2f..... 578
IL_Sgi2fOffset..... 584
IL_SIII_DATA_LENGTH 413
IL_SIII_DATA_TYPE..... 414
IL_SIII_DEVICE_STATUS..... 389
IL_SIII_DEVICE_STATUS_UNION..... 389
IL_SIII_DEVICE_STATUS_UNION_BITS..... 390
IL_SIII_DEVICE_TYPE..... 393
IL_SIII_DIAGNOSTIC_NUMBER..... 391
IL_SIII_DIAGNOSTIC_NUMBER_CLASS..... 393
IL_SIII_DIAGNOSTIC_NUMBER_R..... 392
IL_SIII_DIAGNOSTIC_NUM-
BER_SOURCE_GROUP..... 392
IL_SIII_DIAGNOSTIC_NUM-
BER_SOURCE_TYPE..... 392
IL_SIII_DRIVE_DIAG..... 393
IL_SIII_DRV_ERROR..... 394
IL_SIII_DRV_ERROR_BITS..... 394
IL_SIII_DRV_MESSAGE..... 396
IL_SIII_DRV_MESSAGE_BITS..... 396
IL_SIII_DRV_WARNING..... 395
IL_SIII_DRV_WARNING_BITS..... 395
IL_SIII_ELEMENT..... 414
IL_SIII_IO_DIAG..... 397
IL_SIII_IO_DIAG_STATUS_CODE..... 398
IL_SIII_IO_DIAG_STATUS_CODE_TYPE..... 397
IL_SIII_IoLinkProxy..... 873
IL_SIII_MASTER_STATE_DETAILS..... 385
IL_SIII_PARAM_TYPE..... 414
IL_SIII_REMOTE_SLAVE_STATE_DE-
TAILS..... 385
IL_SIII_REMOTE_SLAVE_STATE_EX..... 388
IL_SIII_REMOTE_SLAVE_STATE_EX_BITS... 388
IL_SIII_SERCOS_PHASE..... 415
IL_SIII_TOPOLOGY_PORT_STATUS..... 390
IL_SIIIAttribute..... 405
IL_SIIICommand..... 341
IL_SIIIElementsToIdn..... 408
IL_SIIIGetConfigSlaveAdrList..... 374
IL_SIIIGetDiagSlaveAdrList..... 376
IL_SIIIGetLiveSlaveAdrList..... 377
IL_SIIIGetSercosPhase..... 409
IL_SIIIMasterState..... 379
IL_SIIIMasterStateDetails..... 380
IL_SIIIParameterBackup..... 362
IL_SIIIParameterRestore..... 368
IL_SIIIReadParameter..... 348
IL_SIIIReadRealParameter..... 352
IL_SIIIReadStringParameter..... 354
IL_SIIIRemoteSlaveState..... 381
IL_SIIIRemoteSlaveStateDetails..... 383
IL_SIIISetSercosPhase..... 410

IL_SIIIStringToIdn..... 412
IL_SIIISvcRead..... 333
IL_SIIISvcWrite..... 338
IL_SIIIWriteParameter..... 356
IL_SIIIWriteRealParameter..... 359
IL_SlaveMapListAddEntry..... 324
IL_SlaveMapListInit..... 323
IL_Ssi..... 652
IL_SsiIn..... 667
IL_Status..... 248
IL_Temp2Rtd..... 589
IL_Temp2Uth..... 584
IL_Temp4_8Rtd..... 595
IL_Temp4_8RtdEf..... 601
Information representation
Names and abbreviations..... 15
InGear..... 18
InOperation..... 18
Inputs and outputs of edge-controlled function blocks, final processing..... 22
Inputs and outputs of edge-triggered function blocks, permanent processing..... 28
Inputs and outputs of state-controlled function blocks, final processing..... 19
Inputs and outputs of state-controlled function clocks, permanent processing..... 25
InSync..... 18
InTorque..... 18
InVelocity..... 18
IoDrvModbusComPort..... 889
IoDrvModbusSerialSlave..... 891
IoDrvModbusTCP..... 890
IoDrvModbusTCPSlave..... 892

K
KNX..... 949
Application example..... 956
Functions..... 949

L
Library
RIH_Dali..... 725
RIH_IndraControl_S20..... 897
RIH_Modbus..... 785
RIH_Safety..... 693
RIH_Safety_ProfibusDP..... 695
RIH_Safety_ProfinetIO..... 707
RIH_Safety_SercosIII..... 719
RIL_Ethercat.library..... 257
RIL_EtherNetIP..... 217
RIL_EtherNetIPAdapter..... 245
RIL_InlineIOModules..... 557
RIL_IoLink..... 871
RIL_MappingList..... 321
RIL_ProfibusDP_02..... 31
RIL_ProfibusDPSlave..... 57
RIL_ProfinetIO..... 69
RIL_ProfinetIODevice..... 205
RIL_SercosIII..... 331
RIL_SercosIIIDeviceReplacement..... 547
RMB_SercosIII_Util..... 417
LinkState..... 210

M
MB_EFC_FUNCTION_CODE..... 481
MB_POWERSUPPLY_ACTUAL_MODE..... 472
MB_POWERSUPPLY_ERROR_MODE..... 473
MB_POWERSUPPLY_FUNCTION..... 475
MB_POWERSUPPLY_MODE..... 472
MB_POWERSUPPLY_TARGET_MODE..... 474
MB_PS_TARGET_MODE_DC_DC_ISLAND_GRID..... 475
MB_SIII_CONTROLWORD..... 444
MB_SIII_DRIVE_SHUTDOWN_MODE..... 445
MB_SIII_DRN_ENABLE..... 545
MB_SIII_DRN_VAR_VEL_ENABLE..... 545
MB_SIII_EFC_CONTROLWORD..... 482
MB_SIII_EFC_STATUSWORD..... 482
MB_SIII_FCP_DUALPUMPSTATUS..... 546
MB_SIII_FCP_MASTERSLAVE..... 545
MB_SIII_FCP_PARAGROUP..... 546
MB_SIII_HOMING_MODE..... 445
MB_SIII_HYDR_CONTROL_MODE..... 526
MB_SIII_IN_PROC_EFC_FREQ..... 481
MB_SIII_IN_PROC_FLOW..... 526
MB_SIII_IN_PROC_POS_INTERPOLATION..... 448
MB_SIII_IN_PROC_POSITIONING..... 448
MB_SIII_IN_PROC_PRESSURE_FLOW..... 527
MB_SIII_IN_PROC_PRESSURE_SPOOL..... 533
MB_SIII_IN_PROC_SPOOL..... 531
MB_SIII_IN_PROC_SWIVEL..... 529
MB_SIII_IN_PROC_SWIVEL_SLAVE..... 530
MB_SIII_IN_PROC_TORQUE..... 447
MB_SIII_IN_PROC_TORQUE_FLOW..... 532
MB_SIII_IN_PROC_TORQUE_SPOOL..... 532
MB_SIII_IN_PROC_VEL_ALT_CTRL..... 534
MB_SIII_IN_PROC_VELOCITY..... 447
MB_SIII_OUT_PROC_EFC_FREQ..... 481
MB_SIII_OUT_PROC_FLOW..... 527
MB_SIII_OUT_PROC_POS_INTERPOLATION..... 448
MB_SIII_OUT_PROC_POSITIONING..... 448
MB_SIII_OUT_PROC_PRESSURE_FLOW..... 528
MB_SIII_OUT_PROC_PRESSURE_SPOOL..... 534
MB_SIII_OUT_PROC_SPOOL..... 531
MB_SIII_OUT_PROC_SWIVEL..... 529
MB_SIII_OUT_PROC_SWIVEL_SLAVE..... 530
MB_SIII_OUT_PROC_TORQUE..... 447
MB_SIII_OUT_PROC_TORQUE_FLOW..... 533
MB_SIII_OUT_PROC_TORQUE_SPOOL..... 532
MB_SIII_OUT_PROC_VEL_ALT_CTRL..... 535
MB_SIII_OUT_PROC_VELOCITY..... 447
MB_SIII_POSCTRLWORD..... 446
MB_SIII_STATUSWORD..... 445

MB_SIIIDRn5020..... 535
MB_SIIIEfcCodeToIdn..... 476
MB_SIIIEfcFrequencyPlcDrive (old)..... 477
MB_SIIIFcP5020..... 541
MB_SIIIPlcDriveDirectValveControl..... 501
MB_SIIIPlcDriveEfcFrequencyType01..... 477
MB_SIIIPlcDriveFlowControl..... 483
MB_SIIIPlcDrivePosInterpolationType01..... 440
MB_SIIIPlcDrivePositioningType01..... 434
MB_SIIIPlcDrivePressureDirectValveControl.... 516
MB_SIIIPlcDrivePressureFlowControl..... 487
MB_SIIIPlcDrivePressureSwivelAngleCon-
trol..... 492
MB_SIIIPlcDrivePressureSwivelAngleSlave..... 497
MB_SIIIPlcDriveTorqueDirectValveControl..... 505
MB_SIIIPlcDriveTorqueFlowControl..... 511
MB_SIIIPlcDriveTorqueType01..... 430
MB_SIIIPlcDriveType01..... 422
MB_SIIIPlcDriveVelocityAltCtrl..... 521
MB_SIIIPlcDriveVelocityType01..... 426
MB_SIIIPositioningPlcDrive (old)..... 440
MB_SIIIPowerSupplyDcDcIslandGrid..... 465
MB_SIIIPowerSupplyDriveIntegrated..... 450
MB_SIIIPowerSupplyStandalone..... 458
MB_SIIIPowerSupplyType03 (obsolete)..... 450
MB_SIIIPowerSupplyType04 (obsolete)..... 458
MB_SIIITorquePlcDrive (old)..... 430
MB_SIIIVelocityPlcDrive (old)..... 426
Modbus RTU..... 889
Libraries..... 889
ModbusRequest..... 893
ModbusServer..... 892
ModbusSlaveComPort..... 889
ModbusTCPSlave..... 890

O

Overview on libraries and target systems..... 2

P

PNIO_CHANNELERROR..... 188
PNIO_DIAGINFO..... 189
PNIO_DIAGTYPE..... 189
PNIO_LIVELIST..... 189
PnsState..... 210
Profinet I/O controller
Special error codes..... 193
PROFINET I/O Device
Special error codes..... 215
Profinet IO controller
Profinet IO status 193

R

RIH_Dali.library..... 725
RIH_IndraControl_S20_KNX..... 949
Approach in case of error..... 952
Diagnostic and error codes..... 950
General information 949
IH_S20KNXClient..... 953
IH_S20KNXDPTxx..... 955
RIH_IndraControl_S20.library..... 897
RIH_Modbus.library..... 785
RIH_Safety_ProfibusDP.library..... 695
RIH_Safety_ProfinetIO.library..... 707
RIH_Safety_SercosIII.library..... 719
RIH_Safety.library..... 693
RIL_Ethercat.library..... 257
Acyclic communication CoE..... 258
Acyclic communication/SoE..... 263
Auxiliary functions..... 301
Data types..... 303
Diagnostics..... 271
Error codes..... 318
Introduction and overview..... 257
State machine..... 295
RIL_EtherNetIP.library..... 217
RIL_EtherNetIPAdapter.library..... 245
RIL_InlineIOModules.library..... 557
RIL_IoLink.library..... 871
RIL_MappingList.library..... 321
RIL_ProfibusDP_02
Several DP master..... 31
RIL_ProfibusDP_02.library..... 31
RIL_ProfibusDPSlave.library..... 57
RIL_ProfinetIO.library..... 69
RIL_ProfinetIODevice.library..... 205
RIL_SercosIII.library..... 331
RIL_SercosIIIDeviceReplacement..... 547
RMB_SercosIII_Util.library..... 417

S

Safety instructions..... 3
Sercos I/O status code..... 398
Service hotline..... 971
Shutdown..... 18
Signal alert symbol..... 4
Signal words..... 4
Special error codes
PROFINET I/O Device..... 215
Profinet IO I/O link device..... 203
Profinet IO I/O link master..... 203
Profinet IO status 193
Standard interfaces at function blocks..... 17
Suggestions..... 16
Support..... 971
Symbols..... 5

W

Warnings..... 3

Notes

rexroth
A Bosch Company

Bosch Rexroth AG
P.O. Box 13 57
97803 Lohr a.Main, Germany
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr a.Main, Germany
Phone +49 9352 18 0
Fax +49 9352 18 8400
www.boschrexroth.com/electrics

R911393275

DOK-IWORKS-FB*LIB**V15-LI05-EN-P