SIEMENS

SIMOTION

SIMOTION Runtime Basic functions

Function Manual

Valid as of version 5.6

02/2024
A5E33425994B

Preface

Fundamental safety instructions 1

System overview 2

Technology Packages and Technology Objects 3

Symbolic assignment (as of V4.2) 4

Programming with Technology Objects 5

Error Handling in Technology Objects 6

Execution System, Tasks, and System Cycle Clocks 7

Programming Execution System/Tasks/System Cycle Clocks 8

Programming of general standard functions 9

Programming of general system function blocks 10

SIMOTION memory concept (in the target device) 11

Downloading data to the target device 12

Error sources and efficient programming 13

Appendix A

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ **DANGER**
indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ **WARNING**
indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ **CAUTION**
indicates that minor personal injury can result if proper precautions are not taken.

NOTICE
indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ **WARNING**
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens Aktiengesellschaft. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens Aktiengesellschaft
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E33425994B
Ⓟ 01/2024 Subject to change

Copyright © Siemens 2024.
All rights reserved

Preface

Content

This document is part of the **System and Function Descriptions documentation package**.

Scope of validity

This manual is valid for SIMOTION SCOUT product version V5.4:

- SIMOTION SCOUT V5.5 (engineering system of the SIMOTION product family),
- SIMOTION Kernel V5.5, V5.4, V5.3, V5.2, V5.1, V4.5, V4.4, V4.3, V4.2, V4.1, V4.0, V3.2
- SIMOTION technology packages CAM, PATH (as of kernel V4.1), CAM_EXT and TControl in the version for the associated kernel.

Sections in this manual

This manual describes the generally applicable functions of SIMOTION and technology objects.

- **System overview**
  General information on SIMOTION.
- **Technology Packages and Technology Objects**
  Basic information on the technology packages and the technology objects.
- **Symbolic assignment**
  Information on symbolic assignment of SINAMICS objects to SIMOTION.
- **Programming of Technology Objects**
  Information on how and with which system functions technology objects can be programmed.
- **Error Handling in Technology Objects**
  General information on technological alarms and command return values.
- **Execution System, Tasks, and System Cycles**
  Information on the execution system and tasks.
- **Programming Execution System / Tasks / System Cycles**
  Information on the programming of the execution system, tasks, and system cycles.
- **Programming of General System Functions**
  Information on which general system functions exist and how they are used.
- **Programming of General System Function Blocks**
  Information which general system function blocks exist and how they are used.
- **SIMOTION Memory Concept (in the target device)**
  Information on the memory concept in the target device.

- **Downloading Data to the Target Device**
  Information on downloading data to the target device or target system.
- **Appendix**
  Information on symbolic constants and identifiers
- **Index**
  Keyword index for locating information

SIMOTION Documentation

An overview of the SIMOTION documentation can be found in the SIMOTION Documentation Overview document.

This documentation is included as electronic documentation in the scope of delivery of SIMOTION SCOUT. It comprises ten documentation packages.

The following documentation packages are available for SIMOTION product version V5.6:

- SIMOTION Engineering System Handling
- SIMOTION System and Function Descriptions
- SIMOTION Service and Diagnostics
- SIMOTION IT
- SIMOTION Programming
- SIMOTION Programming - References
- SIMOTION C
- SIMOTION P
- SIMOTION D
- SIMOTION Supplementary Documentation

Hotline and Internet addresses

SIMOTION at a glance

We have compiled an overview page from our range of information about SIMOTION with the most important information on frequently asked topics - which can be opened with only one click.

Whether beginner or experienced SIMOTION user – the most important downloads, manuals, tutorials, FAQs, application examples, etc. can be found at

https://support.industry.siemens.com/cs/ww/en/view/109480700

Additional information

Click the following link to find information on the following topics:

- Documentation overview
- Additional links to download documents
- Using documentation online (find and search manuals/information)

https://support.industry.siemens.com/cs/ww/en/view/109479653

My Documentation Manager

Click the following link for information on how to compile documentation individually on the basis of Siemens content and how to adapt it for the purpose of your own machine documentation:

https://support.industry.siemens.com/My/ww/en/documentation

Training

Click the following link for information on SITRAIN - Siemens training courses for automation products, systems and solutions:

http://www.siemens.com/sitrain

FAQs

Frequently Asked Questions can be found in SIMOTION Utilities & Applications, which are included in the scope of delivery of SIMOTION SCOUT, and in the Service&Support pages in **Product Support**:

https://support.industry.siemens.com/cs/de/en/ps/14505/faq

Technical support

Country-specific telephone numbers for technical support are provided on the Internet under **Contact**:

https://support.industry.siemens.com/cs/ww/en/sc/2090

Table of contents

Preface **3**

1 Fundamental safety instructions **19**

1.1 General safety instructions 19
1.1.1 Malfunctions of the machine as a result of incorrect or changed parameter settings 19
1.2 Cybersecurity information 19
1.3 Note regarding the general data protection regulation 20
1.4 Danger to life due to software manipulation when using removable storage media 20

2 System overview **21**

2.1 System architecture 21
2.1.1 SIMOTION system architecture 21
2.1.2 SIMOTION SCOUT Engineering System 22
2.1.3 SIMOTION project 23
2.1.4 Offline/online mode 24
2.1.5 Programming 24
2.1.6 Execution system 25
2.2 SIMOTION motion control 26
2.3 Fields of application 27
2.4 Fusion of PLC and motion control 28
2.5 Totally Integrated Automation 28
2.6 Hardware platforms 29

3 Technology Packages and Technology Objects **31**

3.1 Introduction 31
3.2 Technology packages 34
3.3 Reinstalling technology packages 35
3.4 Technology objects (TO) 36
3.4.1 Instantiation and configuration 36
3.4.2 Parameter assignment 37
3.4.3 Programming 37
3.4.4 Interconnections 41
3.4.5 Technology objects and DCC 43
3.4.6 Available technology objects 45
3.5 Reinstalling technology packages via SCOUT interface 46
3.6 Expert list 47
3.6.1 Expert list for configuration data and system variables 47
3.6.2 Using the expert list 50
3.6.3 Comparing expert lists 53
3.6.4 Save expert list comparison 56

3.7 Interconnection of technology objects ... 56
3.7.1 Interconnection overview for technology objects ... 57
3.7.2 Implicit interconnection when creating ... 58
3.7.3 Interconnection using general interconnection screen form in SCOUT (for experts) ... 59
3.7.4 General properties of interconnection interfaces ... 60
3.7.5 Interconnection interfaces on the TOs (for experts) ... 61
3.7.6 Interconnection interface type 'Motion' (for experts) ... 65
3.7.7 Interconnection interface type 'LREAL' (for experts) ... 66

3.8 Technology object trace ... 67
3.8.1 Introduction ... 67
3.8.2 Events ... 69
3.8.2.1 Events ... 69
3.8.2.2 TO trace for PLCopen function blocks ... 69
3.8.2.3 TO trace for commands or function blocks ... 70
3.8.2.4 TO trace for configuration data ... 71
3.8.2.5 TO trace for system variables ... 71
3.8.2.6 TO trace for alarms ... 72
3.8.3 Call ... 72
3.8.4 Parameterization ... 73
3.8.4.1 Overview of the parameterization ... 73
3.8.4.2 Toolbar ... 75
3.8.4.3 Settings ... 75
3.8.4.4 Recorded data ... 76
3.8.4.5 Technology object trace event selection ... 77
3.8.5 Examples/applications ... 81
3.8.5.1 Saving the TO trace recording to the device (memory card) ... 81
3.8.5.2 Reading out the TO trace recording from the device after a reconnection ... 84

4 Symbolic assignment (as of V4.2) ... 87

4.1 Symbolic assignment - introduction ... 87

4.2 Symbolic assignment of TOs ... 89
4.2.1 Assigning the TO axis and TO external encoder ... 89
4.2.2 Assigning the TO measuring input ... 92
4.2.3 Assigning the TO output cam and cam track ... 93
4.2.4 Assigning the TO sensor ... 94

4.3 Symbolic assignment of I/O variables ... 95
4.3.1 Symbolic assignment of I/O variables to the PROFIdrive message frame of the Axis TO ... 95
4.3.2 Interconnecting parameters via BICO ... 96
4.3.3 Assigning I/O variables via the address list ... 100

4.4 Working with the assignment dialog ... 105
4.4.1 Using the assignment dialog ... 105

4.5 Setting up addresses and message frames ... 109
4.5.1 Setting up addresses and message frames - overview ... 109
4.5.2 Setting up communication for symbolic assignment ... 110
4.5.3 Message frame configuration ... 110

4.6 Switching over projects to symbolic assignment ... 114
4.6.1 Working with and without symbolic assignment ... 114
4.6.2 Using symbolic assignment ... 116
4.6.3 Upgrading projects <V4.2 ... 117

4.6.4 Downgrading to versions <V4.2 ... 119
4.6.5 Safety data block for a SINAMICS device version less than V4.x ... 119

5 Programming with Technology Objects ... 121

5.1 Definitions ... 121

5.2 Programming technology objects (TOs) ... 121
5.2.1 Using technology functions in a program ... 121
5.2.2 Sample program with namespace option ... 124
5.2.3 Differences between cyclical and sequential programming ... 125
5.2.4 Function parameters of the technology functions ... 126
5.2.5 Transition and step enabling conditions ... 131
5.2.6 Command execution diagnostics ... 137
5.2.7 Identifiers of technology object instances ... 140
5.2.8 Determining the TO name at runtime (as of V4.4) ... 140
5.2.9 Conversion of TO data types ... 141
5.2.10 System variables ... 143
5.2.11 Configuration data ... 147
5.2.12 Resetting a technology object ... 150
5.2.13 Use of technology packages in libraries ... 151

5.3 Response to faults and events ... 153
5.3.1 Evaluating faults and events ... 153
5.3.2 Execution errors in programs ... 154
5.3.3 Error during operations with floating-point numbers (FPU exceptions) ... 155
5.3.4 Access errors to system variables and configuration data, as well as I/O variables for direct access ... 156
5.3.5 Errors when generating the process image ... 158
5.3.6 Using Taskstartinfo ... 160
5.3.6.1 TaskStartInfo of the StartupTask ... 161
5.3.6.2 TaskStartInfo of the MotionTasks ... 162
5.3.6.3 TaskStartInfo of the BackgroundTask ... 162
5.3.6.4 TaskStartInfo of the TimerInterruptTasks ... 162
5.3.6.5 TaskStartInfo of the UserInterruptTasks ... 163
5.3.6.6 TaskStartInfo of the SynchronousTasks ... 163
5.3.6.7 TaskStartInfo of the ExecutionFaultTask ... 165
5.3.6.8 TaskStartInfo of the PeripheralFaultTask ... 167
5.3.6.9 TaskStartInfo of the TechnologicalFaultTask ... 175
5.3.6.10 TaskStartInfo of the TimeFaultBackgroundTask ... 176
5.3.6.11 TaskStartInfo of the TimeFaultTask ... 177
5.3.6.12 TaskStartInfo of the ShutdownTask ... 177
5.3.6.13 Example for using the TaskStartInfo of the TechnologicalFaultTask ... 178

6 Error Handling in Technology Objects ... 179

6.1 Possible errors in technology objects ... 179

6.2 Process Alarms ... 179
6.2.1 Local response ... 181
6.2.2 Global response ... 181
6.2.3 Error activation ... 182
6.2.4 Configure technological alarms ... 183
6.2.5 Displaying and acknowledging technological alarms ... 185
6.2.6 Acknowledging via the user program ... 186
6.2.7 Evaluating in the user program ... 191

6.3 Return values of commands ... 192
6.4 Errors when accessing system data with _get/_setSafeValue ... 193
7 Execution System, Tasks, and System Cycle Clocks ... 197
7.1 Execution system ... 197
7.1.1 Execution levels / tasks ... 199
7.1.2 Execution system in SIMOTION SCOUT ... 203
7.1.3 Task priorities ... 205
7.1.4 Runtime model in SIMOTION ... 209
7.1.5 Execution system in the symbol browser ... 210
7.1.6 Synchronism of all components ... 211
7.1.7 Identifying the task name and type (as of Kernel V4.5) ... 212
7.2 Description of the user program tasks ... 212
7.2.1 StartupTask ... 212
7.2.2 MotionTasks ... 215
7.2.3 BackgroundTask ... 218
7.2.4 TimerInterruptTasks ... 221
7.2.5 SynchronousTasks ... 224
7.2.6 SystemInterruptTasks ... 232
7.2.7 UserInterruptTasks ... 237
7.2.8 ShutdownTask ... 242
7.3 Configure execution system ... 244
7.3.1 Assigning programs to the execution levels/tasks ... 244
7.3.2 Selecting the cycle clock source ... 249
7.3.3 Defining system cycle clocks ... 250
7.3.4 Assigning system cycle clocks to technology objects ... 257
7.3.5 Task runtimes ... 258
7.3.6 Timeouts and level overflows ... 260
7.3.7 Information on starting a task: TaskStartInfo (TSI) ... 263
7.3.8 Watchdog ... 263
7.4 Second position control cycle clock (Servo_fast) ... 264
7.4.1 Servo_fast (application cycle clock for fast bus system) ... 264
7.4.2 Cycle clock model based on a single position control cycle clock ... 265
7.4.3 Cycle clock model based on two position control cycle clocks ... 266
7.4.4 Cycle clock ratios and conditions with two servo cycle clocks ... 268
7.4.5 Priorities and sequences of synchronous tasks ... 270
7.4.5.1 Cycle clock division when there are two position control cycle clocks ... 270
7.4.5.2 Cycle clock division for the 2nd servo, version 1 ... 271
7.4.5.3 Cycle clock division for the 2nd servo, version 2 ... 272
7.4.6 Synchronization of cycle clock and bus systems ... 273
7.5 Time allocation in the round robin execution level ... 273
7.5.1 Setting of the time allocation ... 276
7.5.2 Settings (examples) ... 277
7.5.3 Processing of the tasks (examples) ... 279
7.6 Task Trace overview ... 283
7.6.1 Using Task Trace ... 283
7.7 Isochronous I/O processing on fieldbus systems ... 283
7.7.1 Data protocol on PROFIBUS DP ... 284

7.7.2 Data protocol on PROFINET IO ... 284
7.7.3 Isochronous data processing ... 285
7.7.4 Dynamic response with respect to data processing in the control ... 289
7.7.5 Dynamic response with respect to data transmission from the acquisition to the processing ... 290
7.7.6 Dynamic response for data acquisition and data output ... 291
7.7.7 Determination of Tdp, Ti and To using HW Config for ET 200 I/O devices on the PROFIBUS... 291
7.7.8 Handling cycle clock scaling ... 293

7.8 Integrating DCC into the SIMOTION execution system ... 294
7.8.1 Introduction ... 294
7.8.2 Sequence model for DCC blocks (DCB) ... 294
7.8.3 servoDccTask in the servo level ... 296
7.8.4 ipoDcc Task in the IPO level ... 297
7.8.5 ipoDcc_2 Task in the IPO2 level ... 298
7.8.6 Execution levels for DccAux and DccAux_2 ... 299
7.8.7 Data exchange between blocks ... 299
7.8.7.1 Data exchange between blocks (overview) ... 299
7.8.7.2 Data exchange between blocks in the same level ... 300
7.8.7.3 Data for blocks from a lower-priority level ... 302
7.8.7.4 Data for blocks from a higher-priority level ... 305
7.8.8 Interconnection of blocks with variables ... 306
7.8.8.1 Interconnection with variables ... 306
7.8.8.2 Behavior for FPU exceptions ... 307

7.9 Include drive I/O ... 308
7.9.1 Assigning the drive I/O symbolically ... 308
7.9.2 Terminal modules TM15 and TM17 High Feature ... 308
7.9.3 Terminal modules TMxx, TB30 and onboard I/Os on SIMOTION D or CX32/CX32-2 and CU3xx/CU3xx-2 ... 308

7.10 Time for SIMOTION and SINAMICS ... 311
7.10.1 SIMOTION time (real-time clock) ... 311
7.10.2 Synchronizing SINAMICS time of day ... 312

8 Programming Execution System/Tasks/System Cycle Clocks ... 317

8.1 Execution system ... 317
8.1.1 Introduction to the execution system ... 317
8.1.2 Execution levels and tasks ... 317
8.1.3 Task start sequence ... 319
8.1.4 Configure execution system ... 320
8.1.4.1 Specifications for the configuring ... 320
8.1.4.2 Assigning programs to the tasks ... 320
8.1.5 Effect of the task execution behavior on the variable initialization ... 321
8.1.5.1 Time for the initialization of local program variables ... 321
8.1.5.2 Assigning initial values to unit variables ... 321
8.1.5.3 Use multiple VAR_GLOBAL, VAR_GLOBAL RETAIN blocks ... 322
8.1.5.4 Influence of the compiler on variable initialization ... 322
8.1.5.5 Marking HMI relevant data ... 325
8.1.6 Task status ... 327
8.1.6.1 Querying and meaning of the task status ... 327
8.1.6.2 Combinations of task statuses ... 328
8.1.6.3 Example of the task status in use ... 328
8.1.7 Waiting in MotionTask for a condition to be satisfied ... 329

8.1.7.1 Syntax of the condition of the EXPRESSION ... 329
8.1.7.2 Syntax of the WAITFORCONDITION statement ... 330
8.1.7.3 Effect of the WAITFORCONDITION statement ... 330
8.1.7.4 Example of use of WAITFORCONDITION ... 331
8.1.7.5 Example for time verification via FB ... 333
8.1.7.6 Example for using WAITFORCONDITION with time monitoring directly in non-cyclic task / Motion Task ... 335
8.1.8 Making tasks wait a defined period ... 337

8.2 Task control commands ... 338
8.2.1 Overview of the task control commands ... 338
8.2.2 TaskId ... 339
8.2.3 Example for using a task control command ... 340
8.2.4 _getStateOfTaskId function ... 341
8.2.5 _resetTaskId function ... 342
8.2.6 _restartTaskId function ... 343
8.2.7 _resumeTaskId function ... 343
8.2.8 _retriggerTaskIdControlTime function ... 344
8.2.9 _startTaskId function ... 345
8.2.10 _suspendTaskId function ... 346
8.2.11 _getTaskId function ... 347
8.2.12 _checkEqualTask function ... 348

8.3 Functions for runtime measurement of tasks ... 349
8.3.1 Functions for runtime measurement of tasks - overview ... 349
8.3.2 _getMaximalTaskIdRunTime function ... 350
8.3.3 _getMinimalTaskIdRunTime function ... 351
8.3.4 _getCurrentTaskIdRunTime function ... 352
8.3.5 Function _getAverageTaskIdRunTime ... 353
8.3.6 Functions for the precise runtime measurement of tasks ... 354
8.3.7 Function _getInternalTimeStamp ... 354
8.3.8 _getTimeDifferenceOfInternalTimeStamps function ... 355

8.4 Functions for message programming (AlarmS) ... 356
8.4.1 General information for the message programming ... 356
8.4.2 AlarmId ... 356
8.4.3 _alarmSId function ... 357
8.4.4 _alarmSqId function ... 359
8.4.5 _alarmScId function ... 362
8.4.6 _getAlarmId function ... 364
8.4.7 Function _getPendingAlarms ... 365
8.4.8 _resetAlarmId function ... 366
8.4.9 _resetAllAlarmId function ... 368
8.4.10 _acknowledgeAlarmSqId function ... 369
8.4.11 _acknowledgeAllAlarmSqId function ... 371

9 Programming of general standard functions ... 373

9.1 Programming of general standard functions - overview ... 373

9.2 Numeric standard functions ... 373
9.2.1 Special features of a numeric function ... 373
9.2.2 General numeric standard functions ... 374
9.2.3 Logarithmic standard functions ... 374
9.2.4 Trigonometric standard functions ... 375

9.2.5 Bit string standard functions ... 376

9.3 Access to bits in bit strings ... 378
9.3.1 _getBit function ... 378
9.3.2 _setBit function ... 379
9.3.3 _toggleBit function ... 382

9.4 Bit operations on numeric data types ... 384

9.5 String processing ... 384
9.5.1 Functions for the string editing ... 384
9.5.2 Error analysis during the string editing ... 386

9.6 Standard functions for data type conversion ... 387
9.6.1 Functions for the conversion of numeric data types and bit data types ... 387
9.6.2 Functions for converting date and time data types ... 392
9.6.3 Functions for the conversion of enumeration data types ... 393
9.6.4 Conversions between BYTE and STRING ... 393
9.6.5 Functions for the conversion of numeric data types to STRING data type ... 394

9.7 Conversion between any data types and defined byte order ... 396
9.7.1 General ... 396
9.7.2 AnyType_to_BigByteArray function, AnyType_to_LittleByteArray function ... 396
9.7.3 BigByteArray_to_AnyType function, LittleByteArray_to_AnyType function ... 398
9.7.4 TO_BIG_ENDIAN function, TO_LITTLE_ENDIAN function ... 401
9.7.5 FROM_BIG_ENDIAN function, FROM_LITTLE_ENDIAN function ... 402

9.8 Combining bit-string data types ... 404
9.8.1 General information for combining bit-string data types ... 404
9.8.2 _BYTE_FROM_8BOOL function ... 404
9.8.3 _WORD_FROM_2BYTE function ... 405
9.8.4 _DWORD_FROM_2WORD function ... 406
9.8.5 _DWORD_FROM_4BYTE function ... 407

9.9 Conversion of technology object data types ... 408
9.9.1 AnyObject_to_Object function ... 408

9.10 Functions for verification of floating-point numbers ... 408
9.10.1 _finite function ... 408
9.10.2 _isNaN function ... 409
9.10.3 IS_VALID function ... 410

9.11 Functions for selection ... 412
9.11.1 SEL function ... 412
9.11.2 MUX function ... 413
9.11.3 MAX function ... 414
9.11.4 MIN function ... 415
9.11.5 LIMIT function ... 416

9.12 Consistent access to global variables of derived data types (UDT) ... 417
9.12.1 General ... 417
9.12.2 _testAndSetSemaphore function ... 418
9.12.3 _releaseSemaphore function ... 419

9.13 Access to system variables and inputs/outputs ... 419
9.13.1 General information on accessing system variables and inputs/outputs ... 419
9.13.2 _getSafeValue function ... 420
9.13.3 _setSafeValue function ... 422

9.13.4 _getInOutByte function ... 426

9.14 Backing up data from the user program ... 427
9.14.1 General information on saving data sets from the user program ... 427
9.14.2 _saveUnitDataSet function ... 428
9.14.3 _loadUnitDataSet function ... 431
9.14.4 _exportUnitDataSet function ... 434
9.14.5 _importUnitDataSet function ... 437
9.14.6 _deleteUnitDataSet function ... 440
9.14.7 _getStateOfUnitDataSetCommand function ... 442
9.14.8 _checkExistingUnitDataSet function ... 443
9.14.9 _deleteAllUnitDataSets function ... 445

9.15 Functions for commandId ... 447
9.15.1 CommandID overview ... 447
9.15.2 _getCommandId function ... 448
9.15.3 _getSyncCommandId function ... 448

9.16 Defining the waiting time ... 450
9.16.1 _waitTime function ... 450

9.17 Device-specific functions ... 451
9.17.1 _getDeviceId function ... 451
9.17.2 _getMemoryCardId function ... 452
9.17.3 _setDeviceErrorLED function ... 453
9.17.4 _setDriveObjectSTW function ... 454

9.18 Determine the memory size of a variable or of a data type ... 456
9.18.1 _sizeOf function ... 456
9.18.2 _firstIndexOf function ... 457
9.18.3 _lastIndexOf function ... 458
9.18.4 _lengthIndexOf function ... 459
9.18.5 LOWER_BOUND function ... 461
9.18.6 UPPER_BOUND function ... 462

9.19 Determination of the logical address for an I/O variable ... 464
9.19.1 Function _getLogicalAdressOfIoVariable ... 464

9.20 Copy areas of arrays ... 465
9.20.1 _ArrayCopy function ... 465

9.21 Display the intermediate results for "program status" ... 467
9.21.1 _trcVal function ... 467

9.22 Creating general references ... 469
9.22.1 REF function ... 469

9.23 Additional available system functions ... 470

9.24 Application of certain system functions ... 471
9.24.1 Programming messages ... 471
9.24.1.1 General ... 471
9.24.1.2 Overview of the functions ... 471
9.24.1.3 Buffer management of AlarmS ... 473
9.24.1.4 Example of message generation ... 474
9.24.1.5 Example of messages with several associated values (as of kernel V4.5) ... 475
9.24.1.6 Checking the error number and status of a message (filtering return values) ... 477
9.24.2 Consistent reading and writing of variables (semaphores) ... 478

9.24.2.1 Consistent data access ... 478
9.24.2.2 Semaphores ... 479
9.24.2.3 Example: Consistent data access with semaphores ... 480
9.24.3 Data backup and initialization from user program ... 481
9.24.3.1 Data backup and data initialization from user program - functions and instructions ... 481
9.24.3.2 Input parameters ... 482
9.24.3.3 Return value ... 483
9.24.3.4 Storage location and memory requirement ... 484
9.24.3.5 Step enabling condition ... 485
9.24.4 Accessing files from a user program (as of V4.4) ... 487
9.24.4.1 Reading and writing files from a user program (as of V4.4) ... 487
9.24.4.2 Managing files from a user program (as of V4.4) ... 497
9.24.5 Converting between any data types and byte arrays (marshalling) ... 501
9.24.6 Communication functions ... 504
9.24.6.1 Available functions ... 504
9.24.6.2 Parameter description for _Xsend ... 505
9.24.6.3 Parameter description for _Xreceive ... 507
9.24.6.4 Parameter description for _GetStateOfXCommand ... 507
9.24.6.5 Communication between SIMOTION and SIMATIC S7 devices ... 508
9.24.6.6 Example of send and receive program ... 510
9.24.6.7 Communication via Ethernet with TCP/IP protocol ... 511
9.24.6.8 Communication via Ethernet with UDP protocol ... 512
9.24.6.9 Acyclic communication with the drive ... 512
9.24.7 Synchronous start ... 514

9.25 HMI (Human Machine Interface) connection ... 519
9.25.1 Interface between HMI and SCOUT or SIMOTION ... 519
9.25.2 Consistent data access with HMI devices (example) ... 521

10 Programming of general system function blocks ... 523

10.1 Overview of the function blocks ... 523

10.2 Bistable elements (set flip-flop) ... 525

10.3 Edge detection ... 528

10.4 Counters ... 530
10.4.1 General information on counters ... 530
10.4.2 CTU up counter ... 530
10.4.3 CTU_DINT up counter ... 531
10.4.4 CTU_UDINT up counter ... 531
10.4.5 CTD down counter ... 532
10.4.6 CTD_DINT down counter ... 532
10.4.7 CTD_UDINT down counter ... 533
10.4.8 CTUD up/down counter ... 534
10.4.9 CTUD_DINT up/down counter ... 534
10.4.10 CTUD_UDINT up/down counter ... 535

10.5 Timers ... 535

10.6 Splitting bit-string data types ... 539
10.6.1 General information for splitting bit-string data types ... 539
10.6.2 _BYTE_TO_8BOOL function block ... 539
10.6.3 _WORD_TO_2BYTE function block ... 540
10.6.4 _DWORD_TO_2WORD function block ... 541

10.6.5 _DWORD_TO_4BYTE function block ... 541
10.7 Emulation of SIMATIC S7 commands ... 542
10.7.1 General ... 542
10.7.2 _S7_COUNTER function block ... 543
10.7.3 _S7_TIMER function block ... 544
11 SIMOTION memory concept (in the target device) ... 547
11.1 Overview of the memory in the target device ... 547
11.2 Memory access ... 550
12 Downloading data to the target device ... 555
12.1 Overview of the data download ... 555
12.2 Save and compile ... 556
12.3 Editing a project with a more recent SCOUT version ... 558
12.4 Performing a consistency check ... 559
12.5 Downloading a project to the target system (all target devices) ... 559
12.6 Downloading CPU/drive unit to target device ... 562
12.7 Downloading a program subset and single units (sources) to the target device ... 564
12.8 Downloading the selected product version of the technology packages ... 566
12.9 Separate initialization of source and TO data during a download ... 567
12.10 Download in RUN ... 567
12.10.1 General information for downloading in RUN ... 567
12.10.2 Download of changed sources in RUN ... 568
12.10.2.1 Change to sources – general ... 568
12.10.2.2 Dependencies on other sources ... 568
12.10.2.3 System unit of the execution system ... 570
12.10.2.4 Changes to the data declaration ... 571
12.10.3 Programmer notes for download of changed sources in RUN ... 571
12.10.4 Error messages if download is not possible ... 572
12.10.5 Recommended settings for download of changed sources in RUN (as of SIMOTION V4.1) ... 573
12.10.5.1 Overview of settings ... 573
12.10.5.2 Compiler option "Only create program instance data once" ... 573
12.10.5.3 Compiler pragma for re-initialization of program instance data for download in RUN ... 574
12.10.6 Initialization of data during a STOP - RUN transition ... 576
12.10.7 Download in RUN before programs in Motion Tasks ... 578
12.10.7.1 Manage MotionTasks via SCOUT (as of V4.1.2) ... 578
12.10.7.2 Switching on debug mode and controlling MotionTasks ... 578
12.10.8 Download without HW Config information ... 579
12.10.9 Download of changed technology objects ... 580
12.10.10 Copying RAM to ROM in RUN (as of V4.2) ... 582
12.11 Download direct to memory card or hard disk ... 583
12.12 Downloading using the device update tool ... 585
12.13 Retaining retentive data during download ... 585
12.14 Loading data from the target device to the programming device (PG)/PC ... 586

12.15 Copy current data to RAM ... 587
12.16 Copy RAM to ROM ... 589
12.17 Deleting user data on the memory card ... 590
13 Error sources and efficient programming ... 591
13.1 Error sources during programming ... 591
13.1.1 Error sources during programming ... 591
13.1.2 Checking data types when assigning arithmetic expressions ... 591
13.1.3 Checking start of functions in cyclic tasks every time ... 591
13.1.4 Wait times in cyclic tasks ... 592
13.1.5 Using the commandId parameter correctly ... 592
13.1.6 Locating errors (ST programs) ... 593
13.1.7 Errors on download ... 594
13.1.8 CPU does not switch to RUN ... 594
13.1.9 CPU goes to STOP ... 595
13.1.10 Checking and setting system clocks ... 596
13.1.11 Comparing REAL or LREAL variables ... 596
13.1.12 Sequential task is interrupted ... 597
13.1.13 Checking for range violations ... 597
13.1.14 Setting size of local data stack ... 597
13.1.15 Sources of errors during multitasking ... 598
13.2 Efficient programming ... 598
13.2.1 Efficient programming - overview ... 598
13.2.2 Runtime-optimized programming ... 599
13.2.2.1 Optimizing access to inputs and outputs ... 599
13.2.2.2 Optimizing access to system variables ... 599
13.2.2.3 Optimal variable declaration ... 599
13.2.2.4 Optimizing access to function block parameters ... 599
13.2.2.5 Optimizing program structure ... 600
13.2.2.6 Optimizing the execution system ... 600
13.2.2.7 Use of USEPACKAGE for multiple technology packages ... 601
13.2.3 Change-optimized programming ... 601
13.2.3.1 Notes on the draft program ... 601
13.2.3.2 Generic programming ... 601
13.2.3.3 Declaring retentive variables in one unit ... 602
13.2.3.4 HMI variables in one unit ... 602
13.2.3.5 Using device global variables versus unit global variables ... 602
13.2.3.6 Centralized starting and resetting of MotionTasks ... 604
A Appendix ... 605
A.1 Symbolic constants ... 605
A.2 Identifiers with defined meaning in SIMOTION ... 607
A.3 Assignment types for symbolic assignment ... 696
A.3.1 Interface types of the technology objects ... 696
A.3.2 SINAMICS interface types ... 698
A.3.3 Interfaces in standard message frames ... 700
A.3.4 Designations for interface types ... 706
A.3.5 Syntax of the assignments ... 708
Index ... 711

Fundamental safety instructions 1

1.1 General safety instructions

⚠ WARNING

Danger to life if the safety instructions and residual risks are not observed

The non-observance of the safety instructions and residual risks stated in the associated hardware documentation can result in accidents with severe injuries or death.

- Observe the safety instructions given in the hardware documentation.
- Consider the residual risks for the risk evaluation.

1.1.1 Malfunctions of the machine as a result of incorrect or changed parameter settings

⚠ WARNING

Malfunctions of the machine as a result of incorrect or changed parameter settings

As a result of incorrect or changed parameterization, machines can malfunction, which in turn can lead to injuries or death.

- Protect the parameterization against unauthorized access.
- Handle possible malfunctions by taking suitable measures, e.g. emergency stop or emergency off.

1.2 Cybersecurity information

Siemens provides products and solutions with industrial cybersecurity functions that support the secure operation of plants, systems, machines, and networks.

In order to protect plants, systems, machines, and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial cybersecurity concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For more information on protective industrial cybersecurity measures for implementation, please visit (https://www.siemens.com/global/en/products/automation/topic-areas/industrial-cybersecurity.html).

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customers' exposure to cyber threats.

To stay informed about product updates at all times, subscribe to the Siemens Industrial Cybersecurity RSS Feed under (https://new.siemens.com/global/en/products/services/cert.html).

1.3 Note regarding the general data protection regulation

Siemens observes the principles of data protection, in particular the principle of data minimization (privacy by design). For this product this means:

The product does not process / store any personal data, only technical functional data (e.g. time stamp). If the user links this data with other data (e.g. shift plans) or stores personal data on the same medium (e.g. hard disk) and thus establishes a personal reference, the user must ensure compliance with data protection regulations.

1.4 Danger to life due to software manipulation when using removable storage media

⚠ WARNING

Danger to life due to software manipulation when using removable storage media

The storage of files on removable storage media involves a high risk of infection, e.g. via viruses or malware. Incorrect parameter assignment can cause machines to malfunction, which can lead to injuries or death.

- Protect the files on removable storage media against harmful software through appropriate protective measures, e.g. virus scanners.

System overview

2

2.1 System architecture

2.1.1 SIMOTION system architecture

The SIMOTION system architecture enables trends towards decentralization, heterogeneous target systems, and distributed intelligence to be implemented.

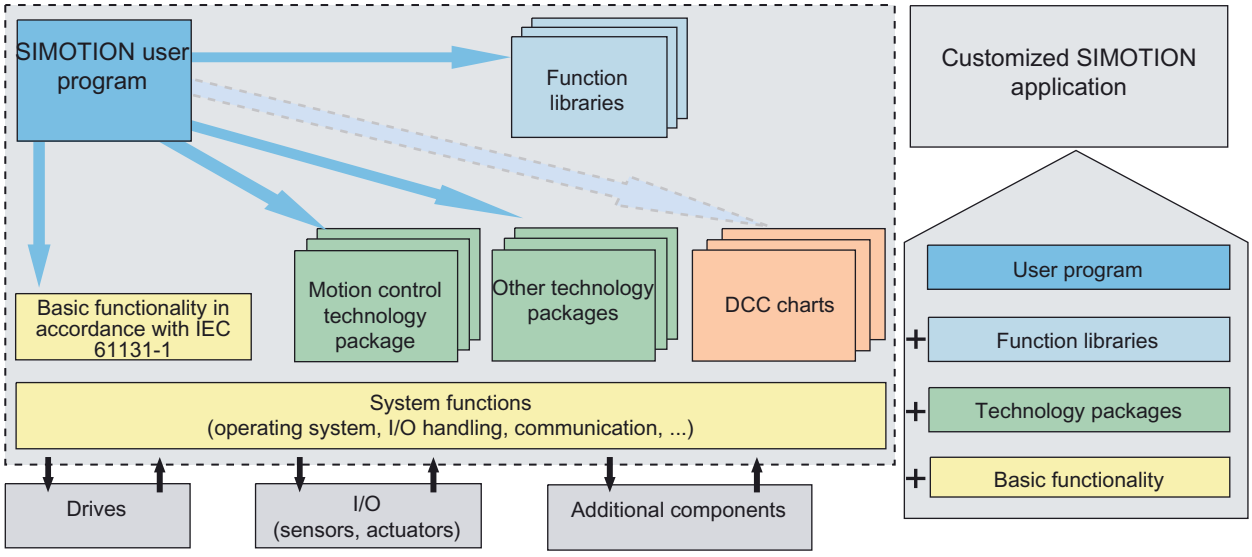


Figure 2-1 SIMOTION system architecture

The basic functionality of the device (SIMOTION Kernel) includes functions for open-loop and closed-loop control as well as logic and arithmetic. Program execution can be cyclical, time-triggered or interrupt-triggered. As a result, the SIMOTION Kernel contains the functions needed for virtually all applications and corresponds in essence to a PLC with the IEC 61131-3 command set plus system functions for controlling various components, such as inputs and outputs.

You can expand the SIMOTION Kernel of the device by downloading technology packages. You access the technology packages from the user program using special language commands, in the same way that you access the SIMOTION Kernel.

For particular tasks, you can either use existing applications or you can program and link the required applications yourself. The applications are programmed in compliance with IEC 61131-3 and can be adapted to your specific task.

In addition, SIMOTION provides function libraries that include the system functions and motion functions. The function libraries contain functions and access to system variables of a technology object and are linked to the associated device and technology package in SIMOTION SCOUT.

For special tasks, such as closed-loop control functions, you can use wiring diagrams and execute them in the corresponding DCC execution tasks using blocks connected with a graphical tool.

2.1.2 SIMOTION SCOUT Engineering System

The following activities are required for machine automation:

- Selection/configuration of the hardware and software components, configuration of hardware and software, including the communication networks with **HW Config**
- Creation and configuration of the technology objects
- Creation of the user program
- Testing and commissioning of the drive units
- Linking of the machine operator control (HMI)
- Concluding steps, such as the generation of machine documentation.

The **SIMOTION SCOUT** engineering system offers a uniform user view and flexible functionality.

Individual automation tasks for production machines are formulated in a uniform and consistent user interface.

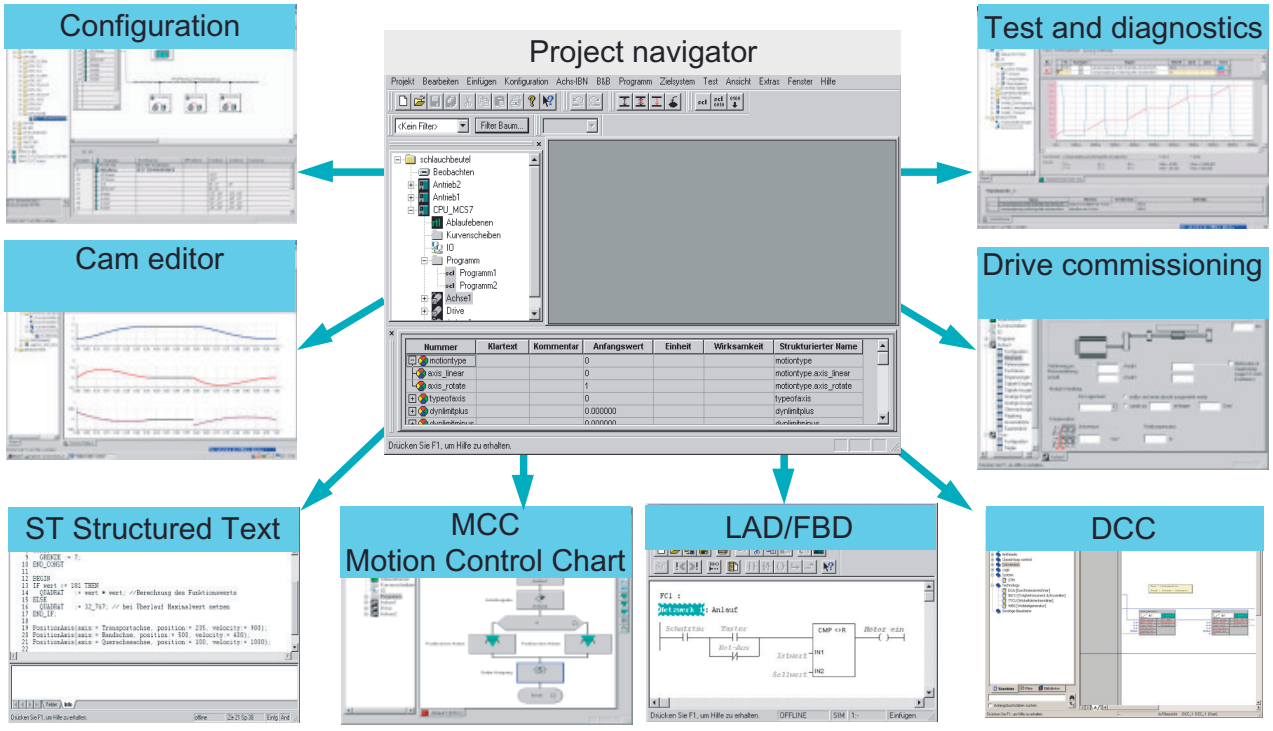


Figure 2-2 SIMOTION SCOUT engineering system

The **SIMOTION SCOUT** engineering system is a powerful tool that acts as the PC development environment to provide optimal support for the required engineering steps in a user-friendly way. It is integrated in the SIMATIC landscape, where it operates as an

option package for **STEP7**. Special attention has been given to optimum usability and a comprehensive, function-oriented view of the automation task.

2.1.3 SIMOTION project

The **project** includes all SIMOTION devices, drives, etc. The associated devices are hierarchically assigned to technology objects and programs. This hierarchical structure is displayed in the **Project navigator**.

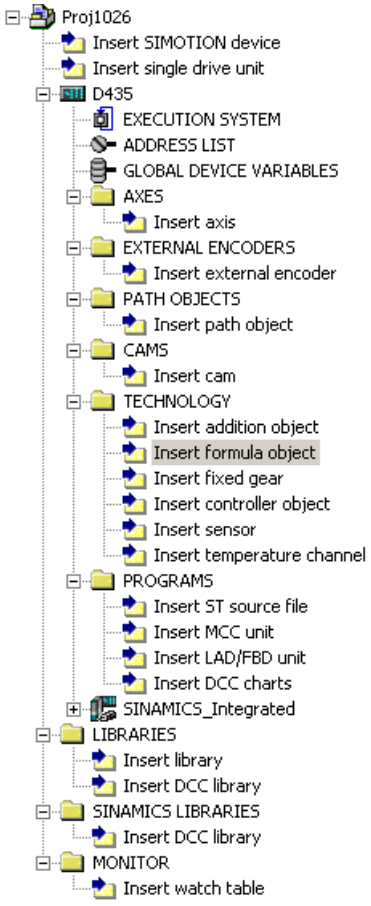


Figure 2-3 SIMOTION SCOUT project navigator

The project is the highest level in the data management hierarchy. SIMOTION SCOUT saves all data which belongs, for example, to a production machine, in the project directory.

2.1.4 Offline/online mode

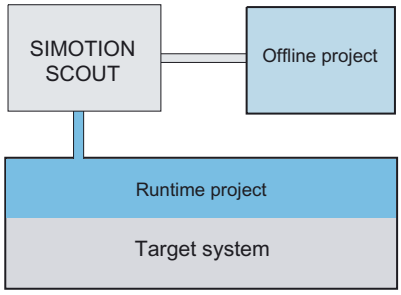


Figure 2-4 Connection between offline and runtime project with SIMOTION SCOUT

SIMOTION SCOUT can be used in two modes:

- In **offline mode**, you can create projects and user programs.
  In offline mode, you work with the SCOUT engineering system without connection to the runtime system. The data and technology objects are maintained in the engineering system. The offline mode is the *leading* work mode, i.e. changes of system variables and configuration data should be made offline.
- In **online mode**, you can load projects and data into the target system, control and monitor the application.
  The data in the SCOUT engineering system is retained as offline project.
  Configuration data changed online can be reloaded into the offline project; this is not possible with system variables changed online.

2.1.5 Programming

The open-loop control and motion control tasks are predefined in the user program.

The following programming languages are available to create your user programs:

- **MCC - Motion Control Chart**
  Is a graphical programming language for programming logic and motion control functions. MCC charts are created.
- **ST - Structured Text**
  Is a text-based programming language compliant with IEC 61131-3 that enables you to create an ST source that can comprise several programs.
  You can also edit an ST source file using an external ST editor.
  Programs created using MCC can be converted to ST, but *not vice versa*.
- **LAD/FBD - Ladder Logic / Function Block Diagram**
  Are graphics-based programming languages in compliance with IEC 61131-3 for programming logic as well as motion control via PLCopen blocks.

- **DCC - Drive Control Chart**
  Is a graphic programming language for programming drive functionality.
  It is possible to edit a Drive Control Chart with an external DCC editor.
- **CLib Studio (C/C++ programming)**
  Is a creation environment in Windows to program functions / function blocks in C/C++ that are stored in user libraries.
  When imported into SIMOTION SCOUT, they can be used in all SIMOTION languages (MCC, ST, LAD/FBD, DCC).

Detailed information for the programming languages is provided in the Programming and Operating Manuals **SIMOTION MCC**, **SIMOTION ST**, **SIMOTION LAD/FBD** and **SINAMICS / SIMOTION DCC editor description**.

Programming in SIMOTION provides the following advantages:

- Programs of different programming languages can be used in one project (**MCC**, **ST** and **LAD/FBD**)
- Programming is independent of the hardware platform
- PLC, motion control and technology can be implemented in one program
- There are functions for direct access to drive parameters available via PROFIDRIVE

Modular machines are supported by the following:

- Modular software development with libraries
- Division into individual machine modules
- Activation/deactivation of DP slaves and technology objects
- Commands for the synchronization

2.1.6 Execution system

SIMOTION provides an execution system with a cyclic task as well as sequential, time-controlled, isochronous, and event-triggered tasks. User programs may be "hooked into" each task; it is even possible for multiple programs to be "hooked into" a task.

Each block is able to use the entire range of SIMOTION functions, i.e. logic, motion, and technology functions.

There is no need to program calls, coordination/synchronization, and monitoring for programs; instead, these functions can be entirely delegated to the system.

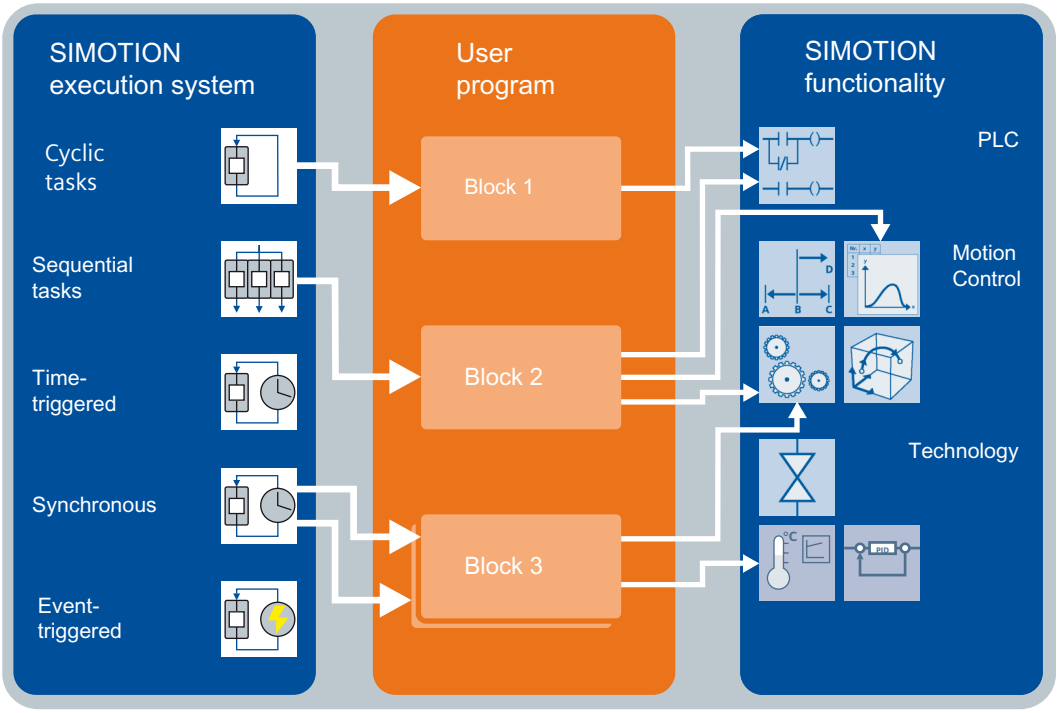


Figure 2-5 SIMOTION execution system

2.2 SIMOTION motion control

SIMOTION offers an optimized system platform for automation and drive solutions with emphasis on motion control applications and technology tasks.

SIMOTION is the answer to the latest trends in production machines:

- **Mechatronics**
  Mechatronics means that a machine is regarded not only from the mechanical aspect, but as a complete system in which mechanical components, electrical components, and control and software technologies are integrated equally. As part of mechatronics, relatively inflexible mechanical components (such as cams, gears, couplings, line shafts, etc.) are replaced by intelligent software solutions.
- **Fusion of PLC functionality and motion control technology**
  The historical separation of pure automation functions and motion functions has been eliminated. These functions are combined both on the hardware and on the software side.
- **Usability**
  A uniform engineering system (SIMOTION SCOUT) ensures consistency in configuration, parameterization and programming. Automation tasks and motion control tasks are programmed in the same language.

- **Standards**
  Industrial automation is increasingly governed by standards in the PC world such as Microsoft Windows and Ethernet. Standardized global programming languages have greatly facilitated system handling for the customer.
- **Modular machine concepts**
  The trend towards standardization is also encompassing machine designs. As a consequence, attempts are being made to break down machine design into various subcomponents. By virtue of this modularity, it is possible to standardize individual subcomponents and install them as *standard* components in different types of machines.

2.3 Fields of application

An efficient control system requires that today's tasks and future trends in the field of production machines are implemented using open control concepts.

SIMOTION is a uniform **motion control system** that focuses on the automation of production machines. The uniformity involves engineering, programming, communication, data management and human machine interface (HMI), and thus encompasses the whole system - even on different hardware platforms.

These trends relate primarily to the following mechanical engineering sectors, for which our motion control system is particularly well-suited:

- Packaging machines
- Plastics processing machines
- Metal forming technology
- Textile machines
- Printing machines
- Machines used in the wood, glass, and ceramics industries
- And other machines

In addition to the standard logic and drive-related tasks, the automation solutions in these sectors are also increasingly requiring the incorporation of integrated, uniform motion control and technology tasks.

The SIMOTION modular system consists of the SCOUT engineering system and a common runtime system for various hardware platforms. The SCOUT engineering system is identical for all hardware platforms. Configuration, parameter assignment, and programming are performed using graphics-based or text-based methods. This also provides the basis for sector-specific solutions.

You decide which runtime software (position, gear, ...) to load to enhance the basic functions based on your application. The fact that SIMOTION runs on a variety of hardware platforms makes it a versatile solution that satisfies all requirements

2.4 Fusion of PLC and motion control

SIMOTION combines open-loop control, technology and motion control. Thus, there are no hardware and software interfaces between the controls required.

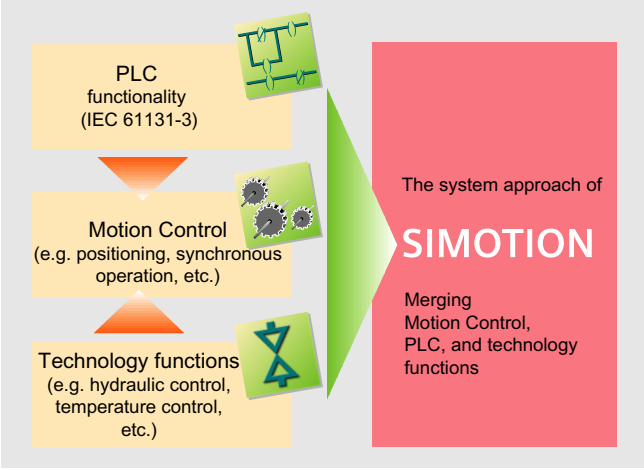


Figure 2-6 Fusion of PLC and motion control

On the hardware side, this means that the programmable controller is able to process motion functions. The hardware platform can be selected individually and hardware components reduced.

On the software side, the fusion of automation functions and motion functions makes for simpler engineering. This starts with the configuration and continues through parameter assignment and programming.

This fusion also means there are no delay times, which results in quick responses and higher cycle rates compared with systems that have separate tasks/CPUs for motion control and the PLC. Not only this, but it is also possible to determine response times more accurately, ensuring high reproducibility of machine behavior and thus of products. In turn, this has a direct impact on product quality.

Consistency with SIMATIC is another essential feature, since both systems can be used together in one plant.

2.5 Totally Integrated Automation

Threefold consistency in programming and configuration, data management, and communication is at the heart of Totally Integrated Automation: In this way, every automation task solved by SIMATIC can also make full use of the many advantages of Totally Integrated Automation:

- Marked reduction in engineering costs
- No system breaks within the automation landscape
- One software basis for all components

It makes no difference whether your automation task involves implementation of customized solutions in machines or systems, or small-scale automation tasks. Totally Integrated Automation with SIMATIC includes all of the technologies that your automation landscape

requires, including programmable controllers, PC-based control, automation computers, distributed I/O, HMI systems, communication networks or process control systems. Because of its modularity, you can use one complete, uniform system to implement the exact solution that satisfies your process requirements and is economically feasible.

SIMOTION and SIMATIC are integrated in the sense of Totally Integrated Automation. This is achieved by integrating SIMOTION SCOUT into SIMATIC Manager and by adopting the same engineering philosophy for comparable activities. Engineering processes that are not available in SIMATIC (motion control, output cam controller, etc.) or that are required by SIMOTION to meet the demands of a distributed system are selected based on optimal usability. Moreover, SIMOTION is integrated into PROFINET and includes all of the requisite system features for this purpose.

2.6 Hardware platforms

SIMOTION supports various hardware platforms. The choice of hardware components is largely driven by your requirements. You can also distribute your automation tasks to different target systems.

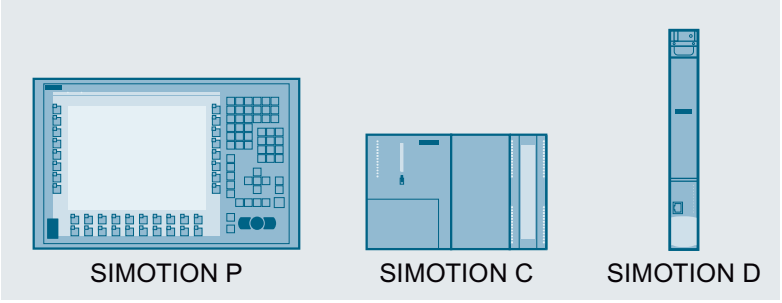


Figure 2-7 SIMOTION hardware platforms

The following platforms are available:

- PC-based (**SIMOTION P**)
  As a PC-based motion control system, SIMOTION P works with the Windows XP operating system, equipped with a real-time expansion for SIMOTION. Apart from SIMOTION applications, other PC applications can also be started.
  SIMOTION P is suitable for:
  - Applications requiring an open architecture to the PC world
  - Applications requiring hardware-based control and visualization
  - Extensive data storage, evaluation and logging
- Controller-based (**SIMOTION C**)
  This controller in SIMATIC S7-300 mounting technology comprises integrated analog drive interfaces and several digital inputs and outputs.
  Moreover, it can be expanded by I/O modules from the SIMATIC S7-300 product range. Two PROFIBUS connections with PROFIdrive interface and one Industrial Ethernet connection allow for communication with other machine parts.
  PROFIBUS can also be used for communication with operator panels (such as those from the SIMATIC HMI range) or with higher-level controls such as SIMATIC S7.
  SIMATIC HMI panels as well as PCs with WinCC flexible or OPC interfaces are suitable as operator systems.
  SIMOTION C enables:
  - Freedom in the selection of the drives
  - Wide range of process signals
  - Retrofit applications via integrated analog interfaces
  - Direct connection of analog drives and stepper drives
- Drive-based (**SIMOTION D**)
  SIMOTION D is a compact, drive-based version of SIMOTION based on the SINAMICS S120 drives family.
  The SIMOTION D Control Units are available in the following variants:
  - SIMOTION D410-2 are compact Control Units for single-axis applications with multi-axis option. The Control Units are available in variants D410-2 DP and D410-2 DP/PN and are snapped onto the SINAMICS S120 PM240-2/PM340 Power Modules in blocksize format. In order to create a multi-axis grouping with SIMOTION D410-2, additional SINAMICS S110/S120 Control Units are connected to the SIMOTION D410-2 by means of PROFIBUS or PROFINET. Motion control is performed centrally by the SIMOTION D410-2 using the SIMOTION technology objects.
  - SIMOTION D4x5-2 are Control Units for multi-axis applications in the SINAMICS S120 booksize format and are available in the four performance variants (D425-2, D435-2, D445-2 and D455-2).
    A SIMOTION D4x5-2 axis grouping comprises a SIMOTION D4x5-2 Control Unit, a Line Module (infeed unit) as well as one or more Motor Modules (power units) to control the motors.
    The fine scalability of SIMOTION D ensures a quick response to changing requirements in automation without having to change the system.

Technology Packages and Technology Objects 3

3.1 Introduction

SIMOTION basic functionality

A SIMOTION system without technology packages and without technology objects provides the execution system and basic functionality of a control system in accordance with IEC 61131-3.

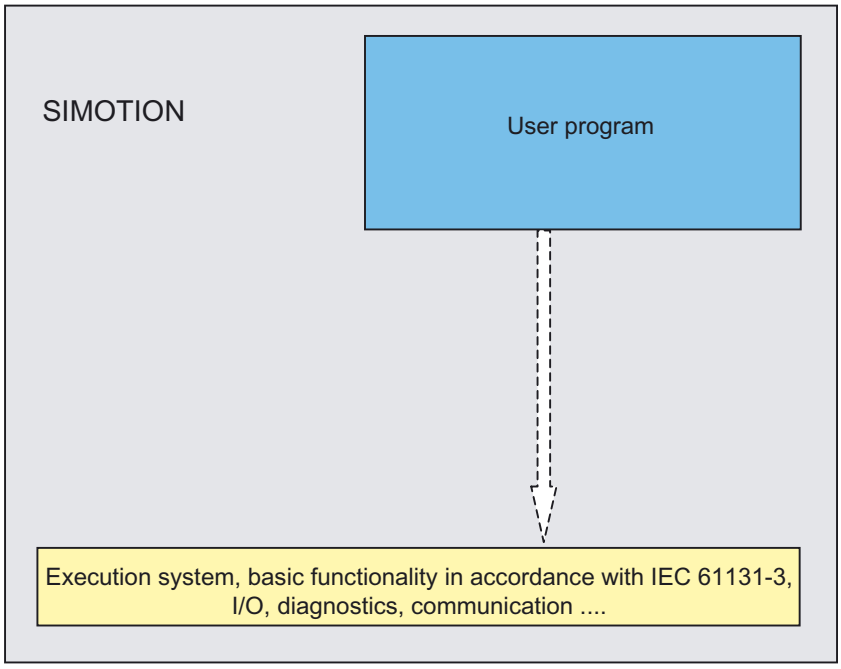


Figure 3-1 SIMOTION basic functionality

The user programs can arbitrarily be assigned to the various tasks.

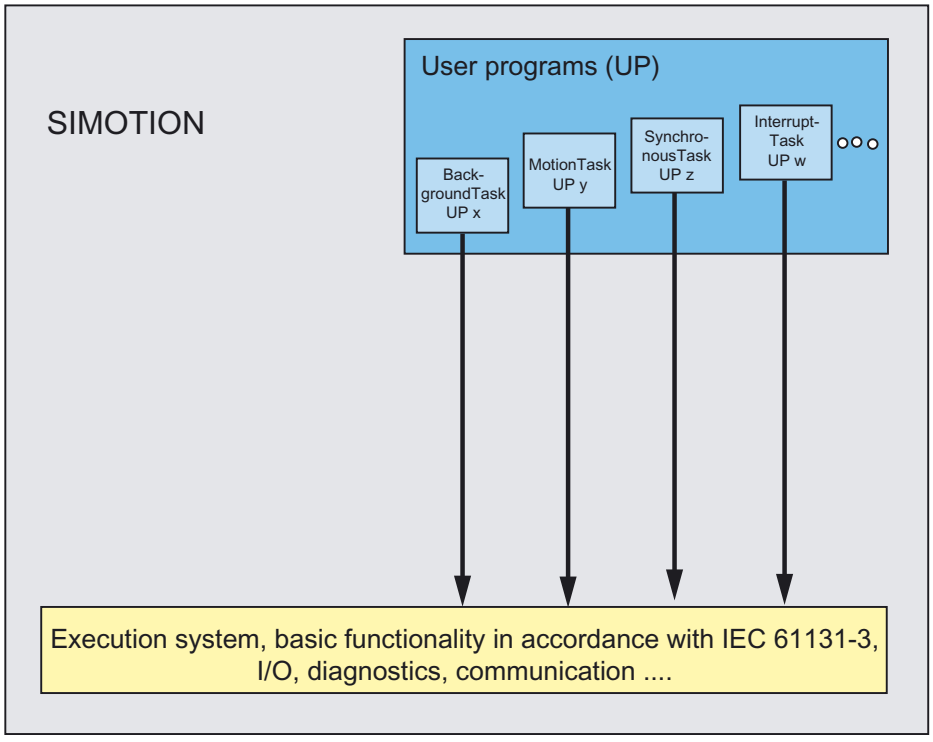


Figure 3-2 Assignment of the user programs to tasks

Technology Objects

The SIMOTION runtime system is implemented in an object-oriented rather than a function-oriented manner, i.e. independent technology objects (TO) are used. These technology objects are available for technology and motion control. Technology objects have a high degree of functionality integrated into them. For example, a TO axis contains the capability for communication with the drive, as well as measured value processing, position control, and positioning functions.

The TOs are created and parameterized by means of configuration and then run automatically in the core of the system. Their functions are called in the user program using appropriate commands. The TOs then process the jobs independently and provide relevant status messages (in a similar way to a driver).

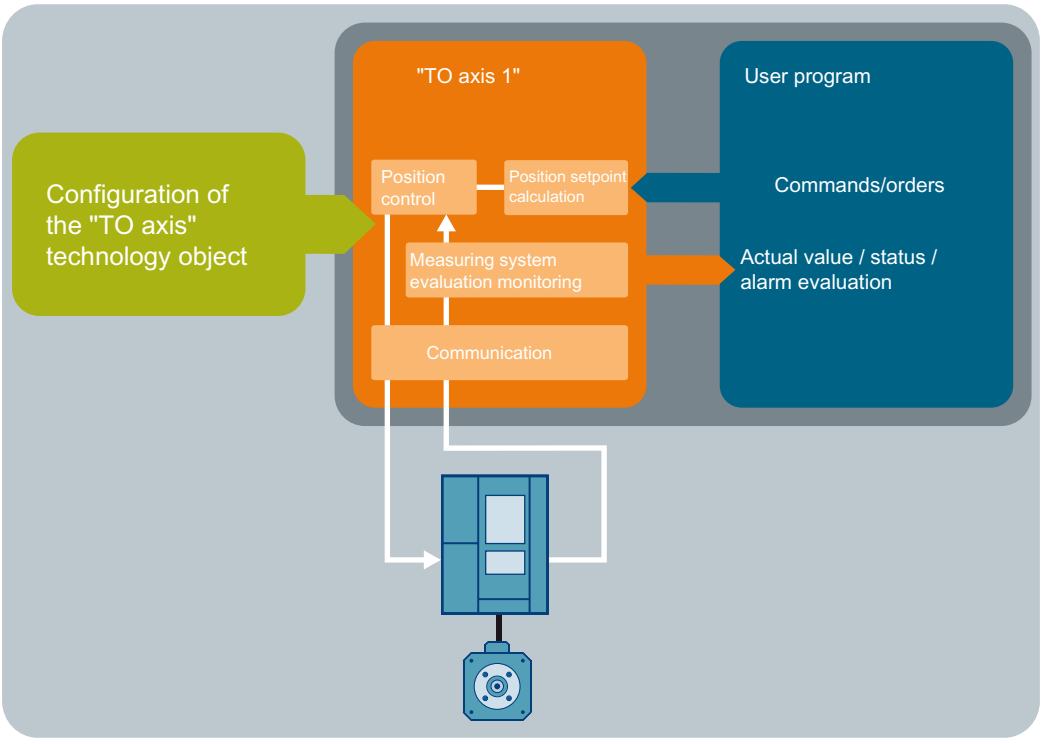


Figure 3-3 SIMOTION technology objects

In principle, there are no limits concerning the number of objects; in other words, you can have any number of axes, synchronous operation links, output cams/cam tracks at an axis, external encoders as master values, etc. The TOs can be activated or deactivated from the program. This enables straightforward adaptation to various machine configurations (e.g. if an axis is not present in a particular configuration).

The technology objects offer the user a technological view of actuators and sensors and provide technological functions for these, for example:

- the TO axis for drive and encoder
- the TO external encoder for one encoder only
- the TO outputCam/camTrack for an output to be switched in a defined way
- the TO measuringInput for a measuring input

Moreover, technology objects for the preparation of technological data on the system level are available, for example:

- the TO FollowingObject for synchronous operation between two axes or one axis on an encoder value
- the TO path for traversing path axes along a path and a position axis synchronous to the path
- the TO cam for representing complex programmable functions
- the TO additionObject, TO formulaObject for the processing of motion data and technology data on the system side

The technology objects are implemented by the system in system tasks, e.g. IPO task, IPO_2 task or servo task.

Instantiation

Technology objects are provided by the system as **technology object types** adapted to the specific application using instantiation.

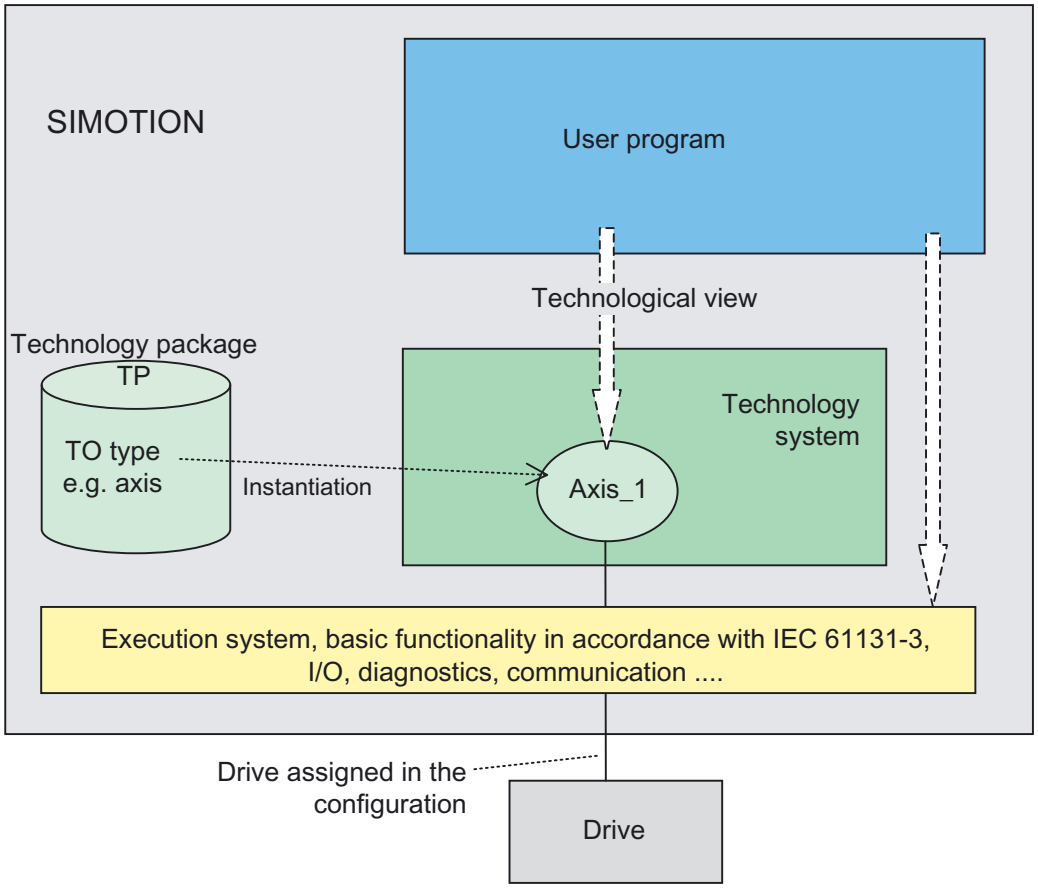


Figure 3-4 Programming model on technology object instances, e.g. on TO axis

Technology object types are combined in one technology package and loaded to the runtime system.

3.2 Technology packages

In SIMOTION, the technology object types are provided via **technology packages**. They are loaded to the SIMOTION runtime system together with the project.

The following technology packages are available:

- **TP CAM** contains the basic technologies for motion control, such as drive axis, position axis, following axis, synchronous object, cam, output cam, cam track, and measuring input.
- **TP Path** contains TP CAM and also the Path technology.
- **TP CAM_ext** contains TP Path and also objects for the preparation of technological data at system level, e.g. addition object, formula object

- **DCC** contains interconnectable blocks for drive-based controller functions.
- **TControl** contains temperature controller technology.

Shell diagram for technology packages

The range of functions offered by the technology packages can be graphically represented in the form of a shell diagram.

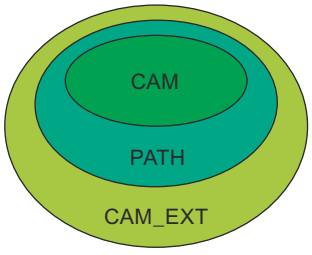


Figure 3-5 Shell diagram for technology packages

Other technology packages

Other sector-specific technology packages are also available as separate products.

See also

Reinstalling technology packages (Page 35)

3.3 Reinstalling technology packages

There are two ways to install and uninstall technology packages:

- via the SCOUT user interface (Page 46)
- via batch files

3.4 Technology objects (TO)

Technology objects / technology object types provide the functionality for technology and motion control, thus comprising technological system functions and hiding the concrete hardware connection.

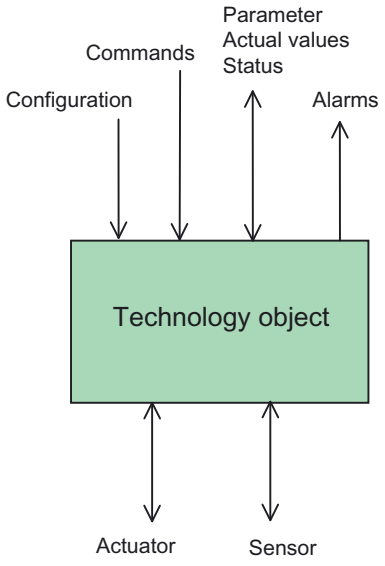


Figure 3-6 Possible interfaces to technology objects

For the cross-TO processing of technological data on the system level, the technology objects provide defined input and output interfaces.

See also

Available technology objects (Page 45)

Interconnections (Page 41)

3.4.1 Instantiation and configuration

With **Add <technology object>**, you create an instance of the corresponding TO type in **SIMOTION SCOUT**. (Data, parameters, alarm lists, etc. are created in the system.)

If necessary, the technology object type must be specified, e.g. for an axis: drive axis, position axis, following axis.

The hardware/software interfaces to the actuator/sensor are defined, e.g. hardware addresses, data width, message frame type for PROFIdrive message frames.

Basic settings are defined, e.g. the processing of the technological system functions in the IPO or IPO_2 cycle clock, see Task runtimes (Page 258).

SIMOTION SCOUT provides wizards and dialogs for the **configuration** of the technology objects. The basic functionality of a technology object is defined by the configuration data.

All entries in the configuration and parameterization screen forms are output in configuration data and system variables, which are displayed in the screen forms as tooltips.

For a detailed description of the configuration data and system variables, please refer to the SIMOTION Reference Lists.

For a description of the procedure of creation and configuration of the individual technology objects, please refer to the respective **function manuals**.

A majority of the configuration data can be modified during the runtime via the user program or the expert list.

See also

Programming of general standard functions - overview (Page 373)

System variables (Page 143)

Configuration data (Page 147)

3.4.2 Parameter assignment

System variables comprise technological parameters and display values of technology objects.

SIMOTION SCOUT usually provides screen forms for the setting of technological parameters and standard values/defaults.

System variables are individual values or structures that are read out consistently.

Additional information on the system variables can be found in System variables (Page 143).

3.4.3 Programming

The technological functions are activated/deactivated using specific commands.

Synchronous/asynchronous command execution

The commands can be set *synchronously* or *asynchronously*.

With the synchronous setting of motion commands, it is possible to wait for a certain motion status or for the end of the motion, e.g. of a positioning, on the command. This setting is especially advantageous when programming motion sequences in sequential tasks (motion tasks).

Return value

The return value supplies the result of the function call, e.g. function was/is carried out as planned or there is an error.

CommandId

CommandId can be used to uniquely identify and trace a TO command.

For information on parameters, step enabling conditions, diagnostics, etc., please refer to the following:

- Functions for CommandID (Page 448) and various examples in this manual.
- **SIMOTION MCC Motion Control Chart**, Programming MCC Charts

For a detailed description of the TO commands, please refer to the SIMOTION Reference Lists.

Programming model

The commands are assigned to tasks which are in turn assigned to the execution levels of the task system.

Commands can be issued from all user program tasks of the system.

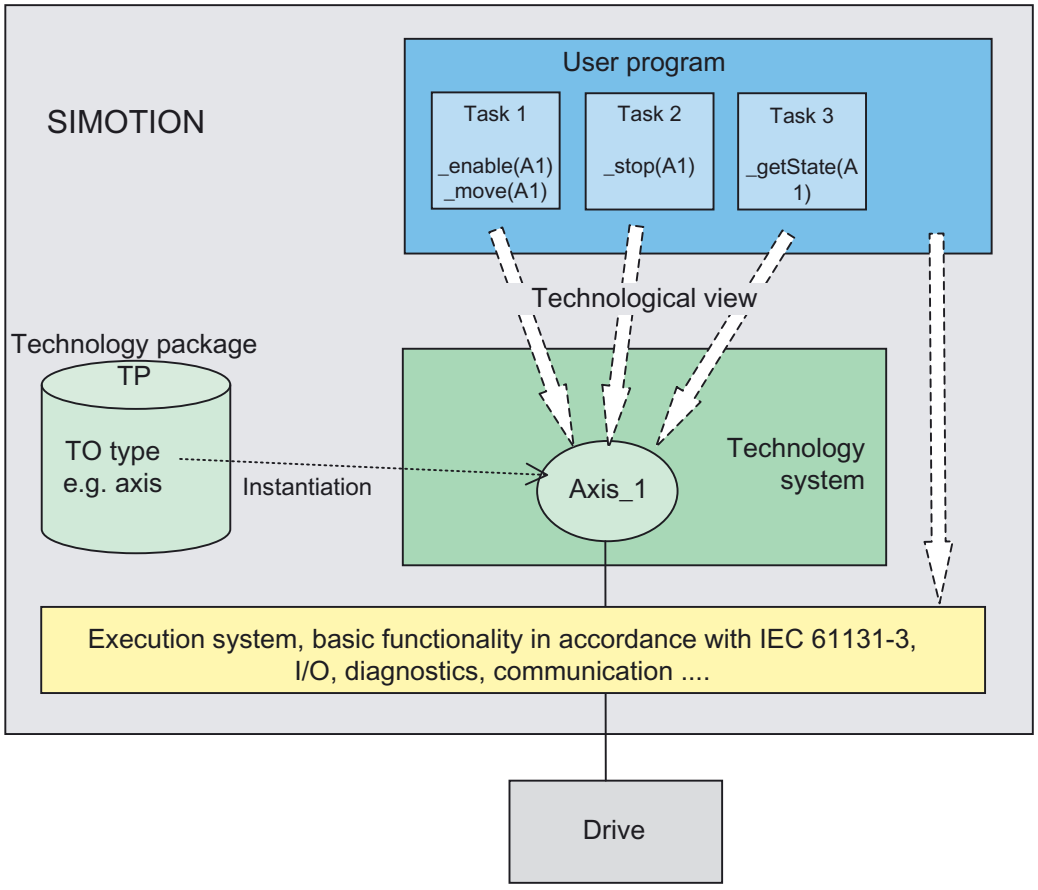


Figure 3-7 Programming model on technology object instances, e.g. on TO axis

The execution time of a command on the technology object is the only factor that determines the effectiveness of the command.

If commands are issued from multiple tasks, programming consistency must be ensured by the user program.

Command execution on the TO / effectiveness of TO commands

The commands issued from the user program, the user program tasks, to the TO can generally be subdivided as follows:

- Commands which are executed immediately in the context/sequence of the user program tasks
  These are handled like a function within the user program tasks.
  These commands are synchronous since the user program is continued only upon the return of the function result.
  Example: Reading TO values
- Commands which are entered in a command buffer and which overwrite each other
- Commands which are entered in a command buffer and which are rejected if the command buffer is occupied

In the TO commands you specify the response of the commands on the TO if more than one command is issued to a command group between two processing cycle clocks, e.g. replace/overwrite or, command rejection.

Information on the command buffer and command activation can be found in the **function manuals** of the individual technology objects.

Execution properties

The execution properties of technology objects are object-specific.

The following synchronous execution levels are provided for the execution of the technology objects: DP cycle clock, servo cycle clock and IPO or IPO_2 cycle clock (except for temperature controllers).

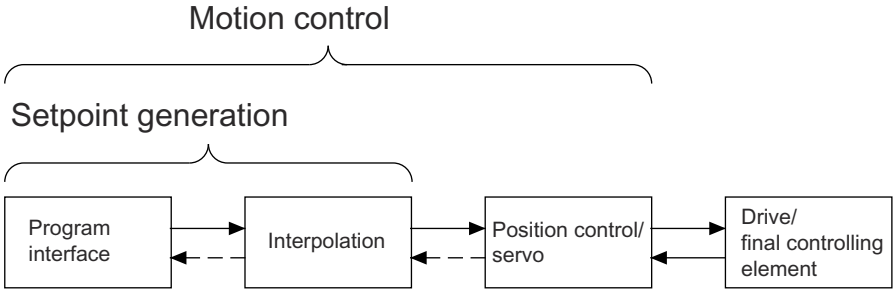


Figure 3-8 Flow chart

The instances of the technology objects are processed in different execution levels.

- The command evaluation and motion control are processed in the IPO/IPO_2 cycle clock.
- The position and setpoint control are processed in the servo cycle clock.
- Communication with the drive is processed via PROFIBUS DP in the DP cycle clock or PROFINET IO with IRT in the PN cycle clock.

The processing can be influenced by means of the system and object configurations.

- System cycle clock setting
  Setting the system cycle clocks also sets the sampling time for the motion control of axes (IPO, IPO_2), the position control (servo), and the communication via PROFIBUS DP or PROFINET IO with IRT.
- When configuring objects, you can specify whether motion control is to be executed in the IPO or IPO_2 cycle clock. This allows you to distinguish between operations that are time critical and those that are not.
- Technology objects are controlled from the user program through system function calls.

With V4.2 and higher, Servo_fast and IPO_fast can also be used as an option in a second application cycle clock.

Alarms

A technology object monitors the execution and executability of technological functions as well as the I/O required for the TO, and creates a **technological alarm**, if necessary.

A **technological alarm** has a TO-local alarm response, e.g. stop motion, and a global response, e.g. stop system or call alarm task in which further responses can be specified. The local and global responses can be set for specific alarms.

For a detailed description of the TO alarms, please refer to the SIMOTION Reference Lists.

See also

Execution system (Page 197)

Differences between cyclical and sequential programming (Page 125)

Process Alarms (Page 179)

3.4.4 Interconnections

The technology objects can be flexibly interconnected (for example, following axes can be switched over or interlinked) and even distributed among several SIMOTION devices. The functionality required for this is in the object and does not need to be programmed. This enables maximum flexibility in terms of functionality and distributability.

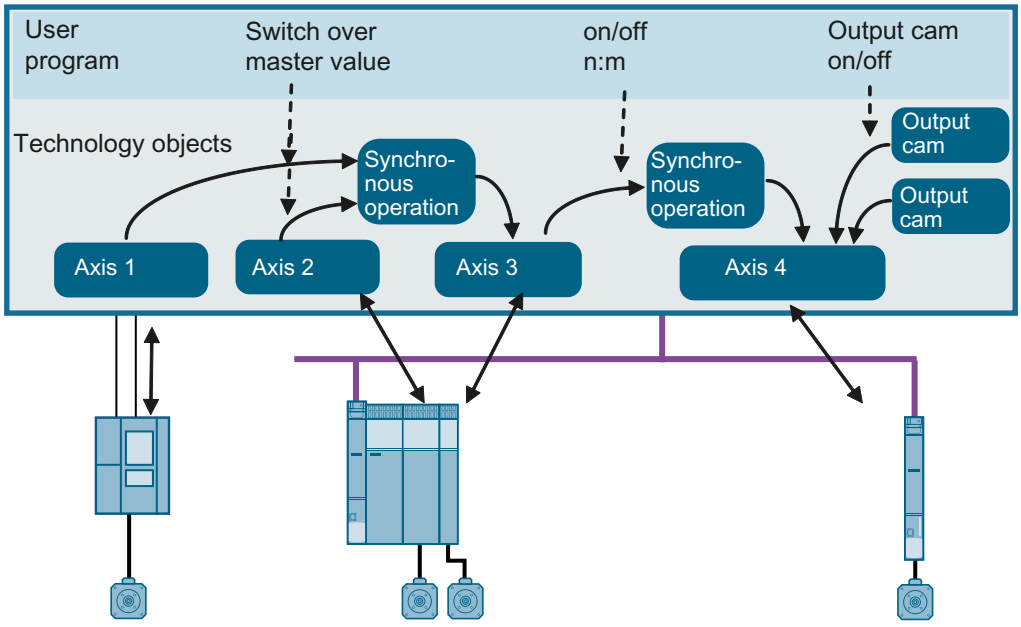


Figure 3-9 Interconnection example

There are interconnection interfaces defined on the technology objects for the exchange of data/information between the technology objects on the system level. These interfaces generally allow for the bidirectional exchange of data between individual technology objects.

The type of the external interfaces can be different on each TO type. There are no interconnections or actuators/sensors required.

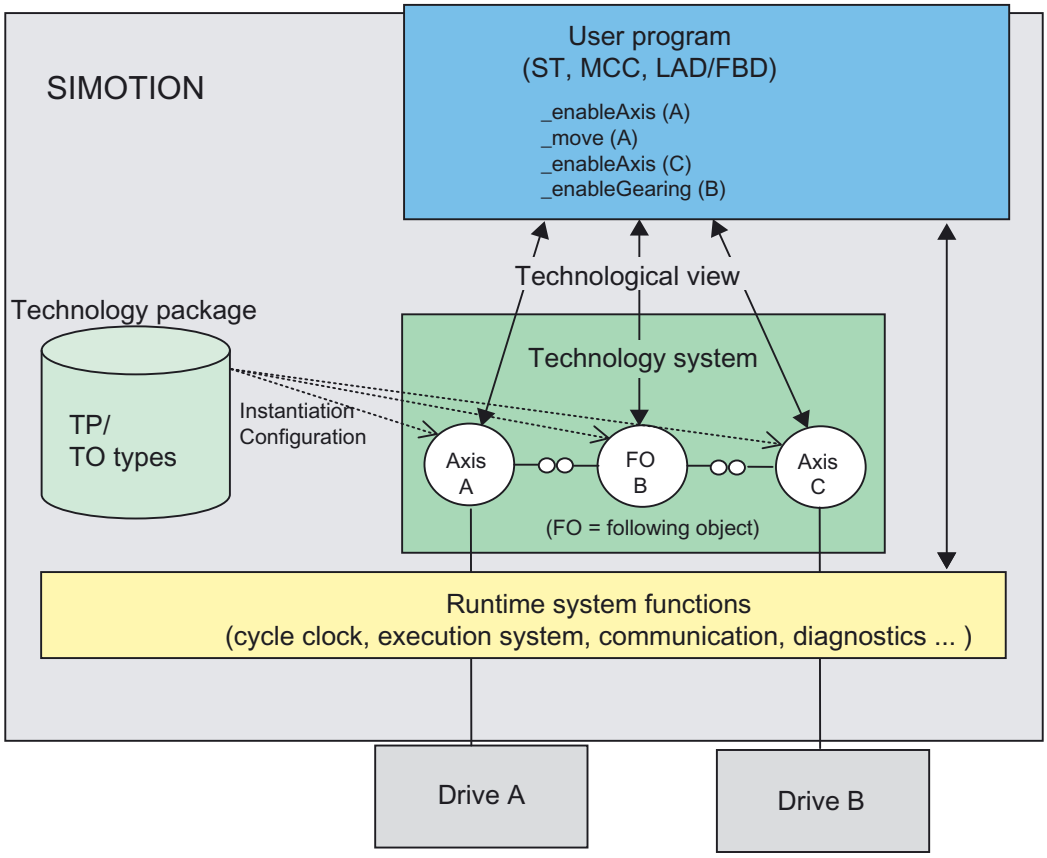


Figure 3-10 Interconnection between technology objects, example: Master axis - Following object - Slave axis

Depending on the interface type, interconnection interfaces can exchange different data in cyclic mode (cyclic motion data, e.g. s,v,a for the motion vector of type motion) and during initialization (e.g. modulo information, units).

Implicit interconnection

If an interconnection is mandatory and unambiguous, it is implicitly created by the SIMOTION SCOUT engineering system, e.g. measuring input with axis, cam with axis, synchronous operation with axis. The corresponding TO types are offered under the axis.

Interconnection using technological interconnection screen forms

There are specific screen forms provided for further interconnections, e.g. for:

- Following axis with master axis/master value
- Following object with cams
- Profile inputs with cams

Interconnection using general interconnection screen form (for experts only)

There is a general interconnection screen form provided for special interconnections, e.g. for:

- MotionIn interface interconnection (for motion vectors)
- Torque input interface interconnection

See also

Interconnection interface type 'Motion' (for experts) (Page 65)

Interconnection interface type 'LREAL' (for experts) (Page 66)

Interconnection of technology objects (Page 56)

3.4.5 Technology objects and DCC

Description

The data exchange between DCC and TO is performed by DCC charts

- using the direct interconnection of block inputs and block outputs with system variables of the TO;
  for TO axes, it is possible to transfer cyclically the data specified in system variables, such as override and setpoints, without commands needing to be issued each time in the user program.
- or by forwarding values calculated in DCC via commands and variables in the user programs to the TO. The TOs have specific function-related interconnection interfaces.

The TOs themselves are not available as blocks in the DCC charts.

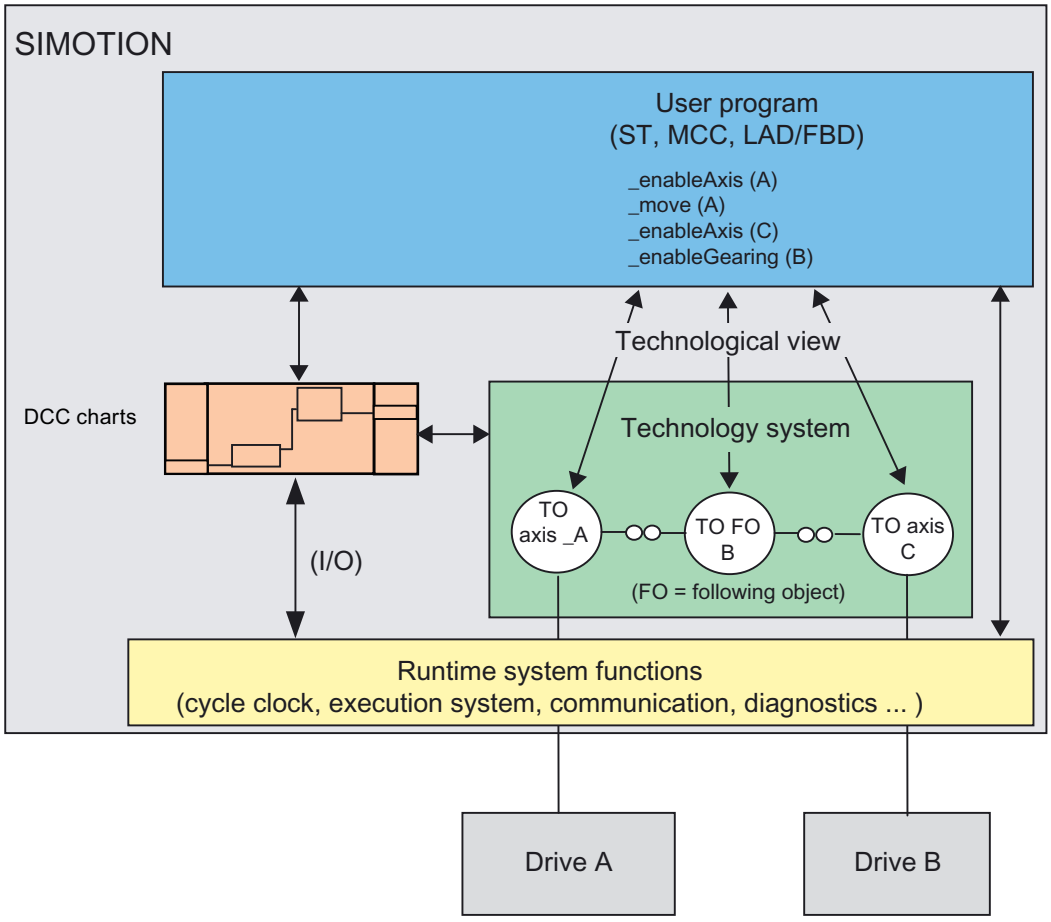


Figure 3-11 Technology and DCC

See also

Sequence model for DCC blocks (DCB) (Page 294)

3.4.6 Available technology objects

Table 3-1 Overview of the technology objects available in SIMOTION

Technology object	Symbol	Brief description
Axis		Axes are available in various versions. There are: • Real or virtual axes • Position axes or drive axes • Electrical axes or hydraulic axes • Standard axes or force/pressure controlled • Modulo axes • Following axes • Path axes For detailed information, refer to the Motion Control TO Axis Electric/Hydraulic, External Encoder Function Manual.
Following object		If an axis is created with the synchronous operation technology, the following object will then be created below the axis. The settings for the synchronous operation are stored in the following object. For detailed information, refer to the Motion Control TO Synchronous Operation, Cam Function Manual.
Path object (as of V4.1)		You can use the Path object technology object to configure the path interpolation. For detailed information, refer to the Motion Control Path Interpolation Function Manual.
Measuring input		Measuring inputs are used for fast, accurate measurement of actual positions. For detailed information, refer to the Motion Control TO for Output Cam and Measuring Input Function Manual.
Output cam		The TO outputCam creates position-dependent switching signals and can be assigned to position axes, following axes or external encoders. For detailed information, refer to the Motion Control TO for Output Cam and Measuring Input Function Manual.
Cam track		Cam tracks are used to group several output cams to form a technology object. For detailed information, refer to the Motion Control TO for Output Cam and Measuring Input Function Manual.
External encoder		The technology object external encoder provides the functionality in the system for connecting an external encoder without an axis, e.g. a shaft encoder on a press. For detailed information, refer to the Motion Control TO Axis Electric/Hydraulic, External Encoder Function Manual.
Cam		The TO cam can be used to define a transmission function and apply it with other technology objects. For detailed information, refer to the Motion Control TO Synchronous Operation, Cam Function Manual.
Temperature channel		The TO temperatureChannel can be used to configure temperature controls in SIMOTION. For detailed information, refer to the Motion Control Additional Technology Objects Function Manual.

Technology object	Symbol	Brief description
Additional technology objects (for experts): The advantage of these technology objects is that they are executed on the system level like other TOs. The interconnected signals are processed directly in the technology objects within a system cycle clock. Applicative solutions in structured text, in contrast, bring about frequent changes between system and application and thus dead times. The additional technology objects are used for the implementation of winder applications, for example.		
Fixed gear		You can use the fixed gear technology object to implement a fixed synchronous operation (without synchronization/desynchronization) using a specified gear ratio. For detailed information, refer to the Motion Control Additional Technology Objects Function Manual.
Addition object		You can use addition objects to add up to four input vectors to produce an output vector. For detailed information, refer to the Motion Control Additional Technology Objects Function Manual.
Formula object		With formula objects, you can use mathematical operations on scalar (LREAL, DINT) and on motion vectors of the Motion type. For detailed information, refer to the Additional Technology Objects Function Manual.
Sensor		The TO sensor can be used to record scalar measured values. For detailed information, refer to the Motion Control Additional Technology Objects Function Manual.
Controller object		The controller object can be used to prepare and control scalar variables. For detailed information, refer to the Motion Control Additional Technology Objects Function Manual.

3.5 Reinstalling technology packages via SCOUT interface

To manage the technology packages using the Scout user interface, open a dialog box under:

Options > Installation of libraries and technology packages

In the dialog box, you can find a list of already installed libraries and technology packages. You control the functionality using the following buttons:

- The 'Add' button enables the installation of new libraries or technology packages. The file manager opens. A *.zip file is to be selected.
- The 'Uninstall' button enables selected libraries to be uninstalled. Uninstalling the technology packages is only possible if you have installed the technology packages and the technology packages are listed in the dialog box. Uninstalling the SIMOTION technology packages is only possible in the shortcut menu of the device.

To subsequently select the newly installed technology package version in the shortcut menu of a device, proceed as follows:

1. To do this, select the device.
2. Open the shortcut menu with a right-click on the device name.
3. Select the "Select technology packages" option:
4. You can select a specific version of the technology package in the "Product version" area of a new dialog box.

3.6 Expert list

In addition to accessing the configuration data and system variables via wizards and parameterization screen forms, you also have the option of direct access via the **expert list**.

Since all important configuration data and system variables can be parameterized via dialogs, the expert list is, however, only required in exceptional cases (such as online modification of configuration data).

These lists state the most important configuration data and system variables for the programming and diagnostics of axes, following object and external encoder.

Starting in V4.1, the expert list has a separate view containing a selection of the most important configuration data and system variables (**Selected Parameters**). This gives you easy access to the most important configuration data and system variables for programming and diagnostics for each individual technology object.

As of V4.2, it is no longer possible to create user-defined lists of values/parameters. The only way to convert existing user-defined lists of values/parameters without encountering any problems is to open them in a watch table with exactly the same range of functions.

Note

Special knowledge of the system is required to modify parameters in the expert list.

- You can overwrite entries made using the expert list by calling wizards and parameterization screen forms.
- No check is made on dependencies with other parameters.

Invoking the expert list

1. Highlight the required technology object in the project navigator.
2. Select **Expert list** in the object context menu or press **Ctrl+E** on the keyboard (if, for example, an axis dialog is open).

Or

- Double-click on the **Expert list** entry in the project navigator below a TO.

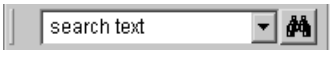
3.6.1 Expert list for configuration data and system variables

The **system variables** and **configuration data** for the technology object are displayed in the first two tabs in the **expert list** .

You can change the configuration data values in OFFLINE mode and in ONLINE mode. Parameters which can be written are displayed in green, those which cannot be written are displayed in yellow. Grayed-out values cannot be changed.

As of V4.2, it is no longer possible to create user-defined lists of values/parameters. The only way to convert existing user-defined lists of values/parameters without encountering any problems is to open them in watch tables with exactly the same range of functions.

Table 3-2 The following information is displayed in the expert list:

Button / Table column	Meaning/Note
Configuration data	The **Configuration data** tab lists the configuration data of the TO in alphabetical order. All configuration data can generally be modified by the user (displayed in green).
System variables	The **System Variables** tab lists the system variables of the TO in alphabetical order. There are system variables which can be written (displayed in green) and which cannot be written (displayed in yellow).
Selected parameters	On the **Selected Parameters** tab, the most important system variables and configuration data of a technology object are displayed in alphabetical order, separated according to system variables and configuration data. The tab can be closed, but it will be displayed again at the next call. The selection is a factory setting and cannot be changed.
Convert list into watch table 	Click this button to convert a user-defined list created using an earlier version to a watch table. A message containing more detailed information appears before the conversion takes place. When you have confirmed the message, you can select the list in a file dialog. Once selected, the list is converted and opens as a watch table.
Quick search 	In this text field, you can carry out a quick search within the expert list. The columns Parameter, Parameter text and Value are searched.
Compare configuration data or system variables 	Click this button if you want to compare configuration data and system variables of 5 technology objects max. and the differences in ONLINE and OFFLINE mode of an object. For further information, see Comparing expert lists (Page 53).
Collect changes 	While the button is activated, all the configuration data changed by you with an effectiveness of "immediately" and "restart" is collected but the new values do not take effect in the target system. The collected changes to the configuration data only take effect in the target system when you click **Activate changes**. See below.
Activate changes 	• If all the collected changes to the configuration data **only** have an effectiveness of **immediately**, the changes take effect as soon as you click **Activate changes**. • If some of the collected changes to the configuration data have an effectiveness of **restart**, **all** changes only take effect after a technology object **restart**.
Activate TO restart 	For configuration data that only take effect after a restart of the TO, the restart can be performed by either activating the system variable "restartActivation" or by clicking the **Restart** button. The button restarts the axis. The button becomes active when you have changed an effective data item after a restart, but not yet accepted it.

Button / Table column		Meaning/Note
Configuration selection list		Select whether the parameters for a linear axis or rotary axis are to be displayed or changed for standard, force or pressure in the table in the expert list.
Parameter 		The name of the configuration data item or the system variable is displayed here. Click the plus sign to open the entire structure.
Parameter text		A short description of the system variable or the configuration data is displayed.
Offline value		The value of the configuration data item or the system variable is displayed here. Depending on the type of parameter, you can enter the value directly as a numerical value or select a symbolic identifier from the selection list.
Configured value	(ONLINE only, configuration data)	The configured value of the configuration data item is displayed here. This value is saved to the RAM of the target device.
Current value	(ONLINE only)	The current value of the system variable or the configuration data is displayed here. You can change the value of the system variable directly in ONLINE mode. Changes to the configuration data can only be made in the **Next value** column. Current values are stored in the **current data memory**. To transfer these values to the RAM, **Target system > Copy current data to RAM** must first be selected in the menu. **Note:** System variables changed ONLINE cannot be copied to RAM. This also means that it is not possible to **Save to memory card (Ram2Rom)** or **"Save in the engineering project (load configuration data in PG)**. See SIMOTION memory concept.
Next value	(ONLINE only, configuration data)	You can enter the **Next value** of the configuration data item in ONLINE mode here. This value is then taken over as Current value depending on when the configuration data item is to take effect. With Immediately, the new value takes immediate effect in the target system after pressing the ENTER key or selecting a value.
Units		The unit of the system variable or the configuration data is displayed.
Effectiveness		This displays when the value of the changed configuration data item takes effect, e.g. with **restart**, the changed value takes effect in the target system after a restart. You cannot change certain configuration data in ONLINE mode. With V4.1.2 and higher, this column is also displayed OFFLINE.
Data type		The **data type** of the system variable or the configuration data item is displayed here (e.g. INT for an integer value).
Minimum		The minimum value of the system variable or the configuration data item is displayed.
Maximum		The maximum value of the system variable or the configuration data item is displayed.
Filter		This line allows you to enter filtering criteria for the contents of the various columns in the table. Alternatively, you can select predefined filters in the selection box. To reset the filters, select the criterion **All** in the selection box or click the  symbol.

See also

Comparing expert lists (Page 53)

3.6.2 Using the expert list

Displaying the help for the system variables and the configuration data

1. Select **Help > Context help**.
   Or
   Press the **SHIFT+F1** keys.
   Or
   Select **Parameter help** in the context menu.
   The cursor changes to a question mark.
2. Click the configuration data item or the system variable in the expert list for which help is required.
   The help for this is displayed.

Changing the data in the expert list offline

1. In the expert list, select the Configuration data or System variables tab.
   A tabular overview is displayed.
2. In the **Parameter** column of the expert list, click the plus sign in front of an entry to display subentries.
   You cannot change entries highlighted in yellow.
3. Click the **Offline value** column of the entry to be changed.
4. Enter the value directly or select a value when a selection list appears.

Carrying out a quick search

When entering a search term, the cursor jumps to the first line in which the search term is found.

- Enter the search term and press the ENTER key.

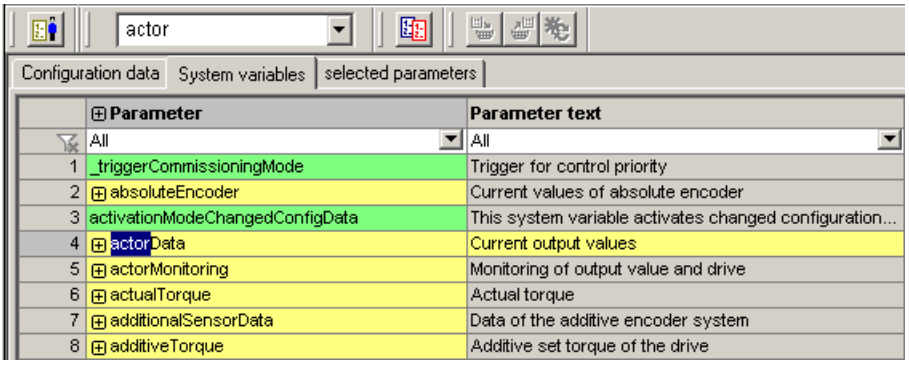


Figure 3-12 Example of quick search

If it is a writable parameter, the cursor jumps directly to the value field.

If a structure shell is found that does not have a separate parameter value, the jump is made directly to the cell in which the term was found.

Continue search:

- Press the **F3** key.
  The next cell containing this term is searched.

If the search function does not find any further entry in the list, a message is output.

Navigating within the expert list using the arrow keys

Each cell within the expert list can be selected using the arrow keys.

1. Press an arrow key.
   The cell located next is selected.
2. With the **Ctrl+arrow** key combination, the cursor jumps to the end or beginning of the respective line or column.

Selecting cells using the keyboard

If you want to select the complete expert list, including the cells in the non-visible area:

- Press the **Ctrl+A** key combination.

In online mode, this key combination also initiates a read process for all of the parameter values (with technology objects as of V4.0).

Individual cell areas can be selected by navigating with the arrow keys or with the mouse:

- The cells are selected in the course of navigating by simultaneously pressing the **Shift** key.

Cell areas can also be selected using the mouse:

- Place the cursor on a line, press the left mouse button and drag the mouse pointer across the cell area to be selected.

Copying cell contents into other Windows applications

Selected rows or cells can be copied into other Windows applications using the Windows clipboard:

- Press the **Ctrl+C** key combination or select **Copy** in the context menu.

From the clipboard, the cell content can be inserted, e.g. in an Excel or Word document.

How to create a watch table

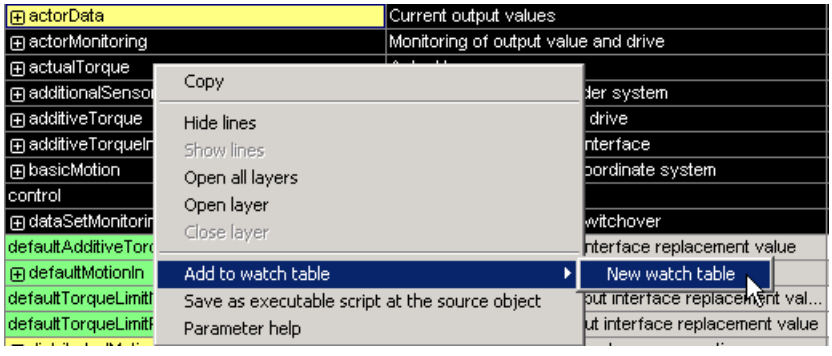


Figure 3-13 Context menu in the expert list (right-click)

Table 3-3 Context menu in the expert list

Menu command		Meaning/function
Copy		Copies the parameters selected
Hide lines		Hides the marked lines.
Show lines		Shows the hidden lines again.
Open all levels		Opens all levels
Open level		Opens all the levels selected
Close level		Closes any levels which are open
Add to watch table		
	New watch table	Creates a new watch table and opens it in the **<Watch table name>** tab of the detail view
	Watch table name	Adds parameters to an existing watch table
Add to watch table, values as control values		
	New watch table	Creates a new watch table and opens it in the **<Watch table name>** tab of the detail view
	Watch table name	Adds parameters to an existing watch table
Save as executable script at the source object		Saves as executable script
Parameter help		Calls the help for the parameter selected

Procedure for creating a watch table

1. Select the lines in the expert list to be transferred to the watch table.
2. In the context menu (right-click), select **Add to watch table > New watch table**. The **Insert Watch Table** dialog opens.

   Note

   To add to an existing watch table, select **Add to watch table > <Watch table name>**. No more dialogs are called. The values are added to the existing watch table.

3. In the **Insert Watch Table** dialog, you can enter the name, author, version, and comment. The name is preassigned automatically.

Note

You can also save the results of an expert list comparison as a watch table or as an executable script, see Comparing expert lists (Page 53).

Notes on the structure of scripts:

- Each converted script receives a header with date and origin, and then an empty line.
- The logging mechanism of the script engine is activated in the next line. This causes each write and read action to be output in the script output window.
- Headings in the watch table are entered in the script as comments with a leading additional empty line.
- Comments in the watch table are entered in the script as simple comments without an additional empty line.

- For each write parameter in the watch table, a write request is entered in the script. Numbers and enumerations are written as integer values. The parameter text of the parameter is added as comment at the right-hand side next to the write request.
- Read-only parameters in the watch table are no longer saved in scripts as of V4.2. When parameters are saved as a watch table, only the parameter (without a value) is added to the watch table.

Example for the use of scripts:

- Any changed unit systems will not be used when the script is executed. You must yourself ensure that the acceptance of the parameter values is sensible in the current context.
- Parameters that change the structure (e.g. TypeOfAxis.typeOfAxis) are not treated differently. You must yourself ensure that these parameters are at the top of the parameter selection.
- Only the current value is written; this means the script may not be executed online if it contains configuration data.

Application: You can also transfer the parameter assignment from one axis to the other with scripts.

3.6.3 Comparing expert lists

Description

In this window, you can compare the differences in parameter settings of a maximum of five objects (TOs) and the differences in ONLINE and OFFLINE mode of an object.

The displayed data is always the current data in effect. For TOs in ONLINE mode, this data is the actual values. System variables are not permanently updated, only if you update the representation with **Start comparison**.

You can change the values of the comparison objects as well as the values of the reference objects. Objects that cannot be changed are indicated with "???".

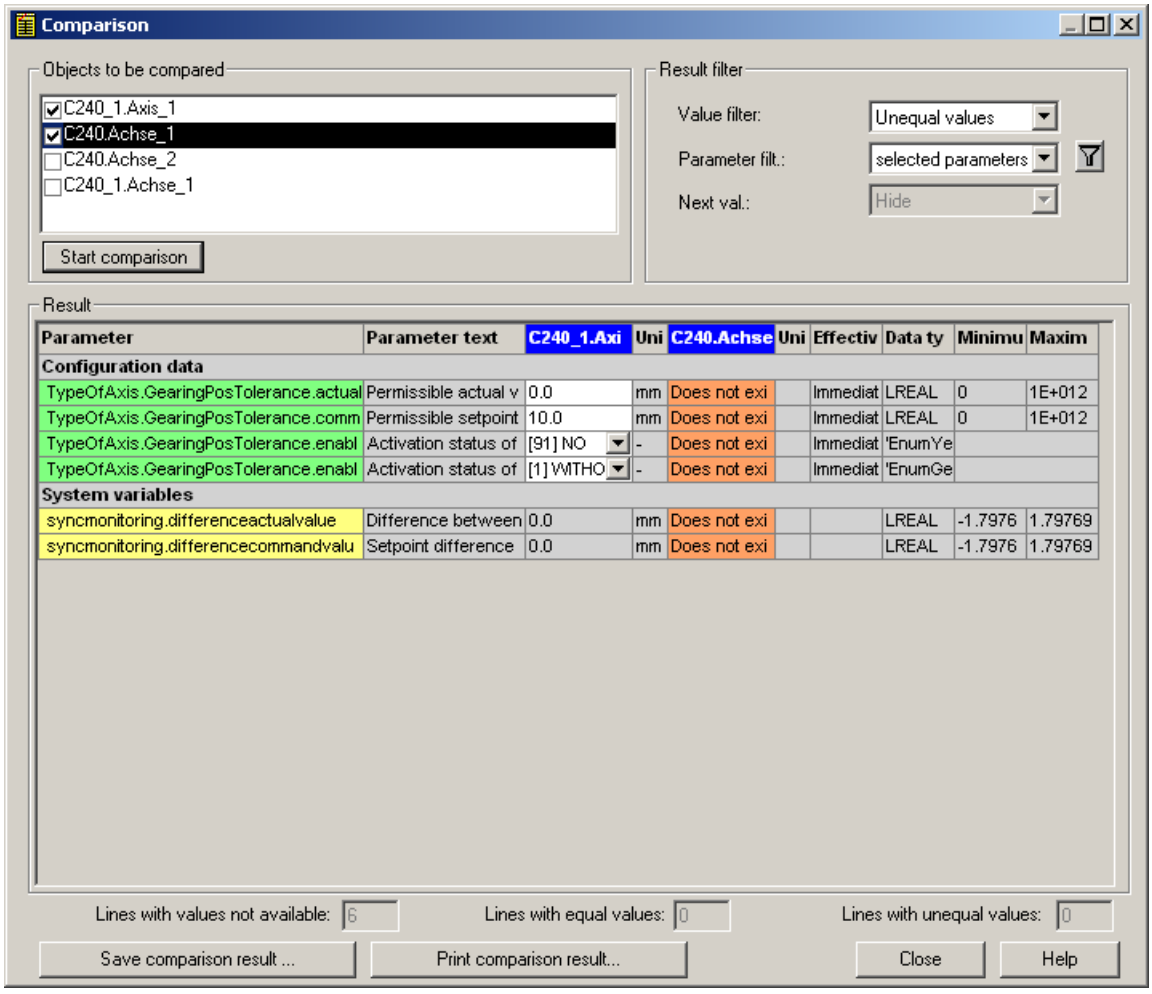


Figure 3-14 Experts list comparison

Parameters with different settings or parameters not available for an object are presented in a list.

Note

A comparison can only be made within groups of compatible objects (TOs). Only the eligible objects for the comparison will be made available.

Button / table column	Meaning/information
Objects to be compared	Select the checkbox for the objects you want to compare. You can compare up to five objects with one another and objects in OFFLINE and ONLINE mode. In ONLINE mode, separate checkboxes are available for ONLINE and OFFLINE parameterization of an object. The uppermost list, which is grayed out, is used as a reference list. The display is always dictated by the structure and, thus, the parameter set of the reference object. As a result, for comparison objects it can only be indicated whether or not they have the parameter of the reference object. Parameters that only the comparison objects have but not the reference object are not displayed. If the values of a setting are different, this is indicated by the appearance of a light-orange colored background in the relevant cell in the columns of the relevant comparison objects. However, the relevant cell of the reference object does not display a colored background.
Start comparison	Click **Start comparison** to compare the parameter settings of the selected objects. The compared settings are listed under **Result**.
Value filter	Select the parameter types that are to be displayed under **Result** after the comparison.
All	All the parameters that can be compared are displayed under **Result**.
Unequal	Only the parameters whose values are unequal will be displayed under **Result**.
Equal	Only the parameters whose values are equal will be displayed under **Result**.
Parameter filter	Only the type of parameters are displayed which you select in the drop-down list.
Display filter	Click the button to open the **Display Filter** window. Here you can edit the display of the columns.
Next value	The next values (values in the next memory) are also displayed.
Result	**Result** displays the parameters found to have different settings during comparison of the objects and other parameters that are not available for an object. Diverging values and non-comparable values of parameters are displayed with a colored background.
Save comparison result	Click the button to save the comparison results as a new watch table or as an executable script. The button is active only if the parameters of a device are displayed under **Result**. For further information, see Save expert list comparison (Page 56).
Print comparison result	Click the button to print the comparison results in a tabular form. The parameter number, the value for the respective object and the unit will be printed in each case.

See also

Save expert list comparison (Page 56)

3.6.4 Save expert list comparison

Description

In this window, you can save the result of the comparison.

The button is active only if the parameters of a device are displayed under Result.

- Click **As watch table** to create a new object-granular watch table. The left column is always the reference column, i.e. only the parameters of the reference object are compared and saved.
- Click **Executable script on source object** and select the object to which you want to assign the script. A SCRIPTS folder will then be created under this object.

See also Comparing expert lists (Page 53).

You can save only the comparison results of one object. Prior to storing, you must specify which object is to be saved.

3.7 Interconnection of technology objects

Technology objects are automatically interconnected when they are created (synchronous axis, output cam, measuring input). Additional interconnections can be established using a general interconnection screen form.

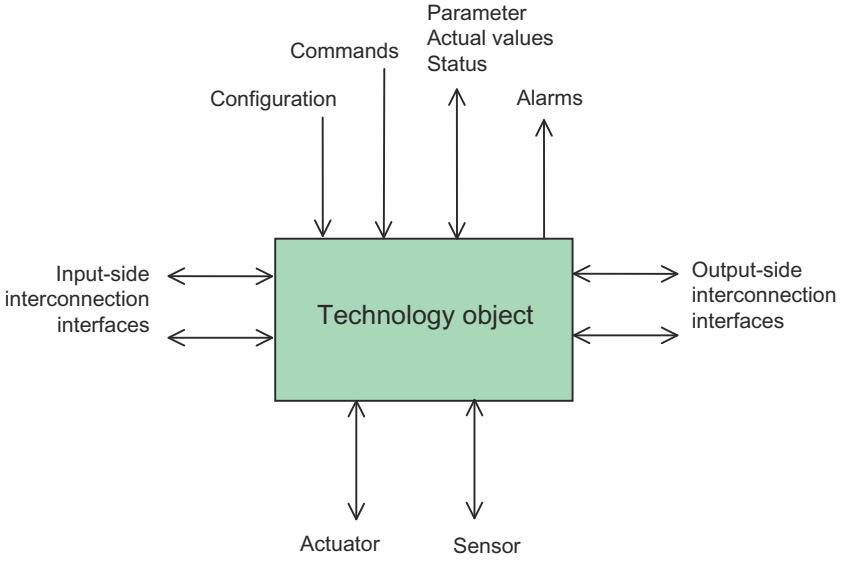


Figure 3-15 Interconnection interfaces to technology objects

3.7.1 Interconnection overview for technology objects

Introduction

With the interconnection overview you can display all motion input and output interconnections of technology objects in the project via an interconnection tree. The tree display enables the interconnections to be displayed in cascades.

Note

As of V4.3, it is possible to assign device names that do not comply with the ST naming convention. Names with special characters such as "." are displayed in the interconnection overview in " ".

An interconnection table shows the technology objects interconnected on the input or output side with interface designation for the technology object selected in the interconnection tree.

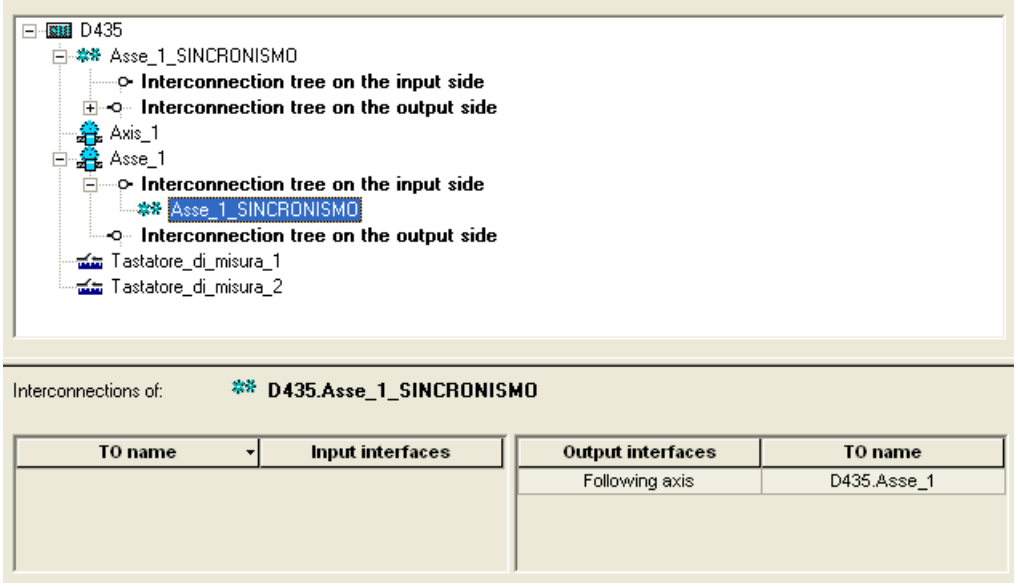


Figure 3-16 Interconnection overview

Properties of the interconnection overview

The interconnection overview can be started with the button or via **Edit > Interconnection overview**.

- After the interconnection overview has been started, the device level (CPUs) is displayed first. You can then display all instantiated TOs under each CPU by opening the appropriate interconnection tree of the CPU. The TOs are sorted according to TO type and displayed in alphabetical order within a TO type.
- Two interconnection trees are offered for the TOs that are displayed directly below the CPUs:
  - Input-side interconnection tree; indicates the motion input interconnection
  - Output-side interconnection tree; indicates the motion output interconnection
- All technology objects that have motion inputs and outputs interconnected to other technology objects, e.g axes, synchronous objects or encoders, can be opened further. This is possible at every hierarchy level of the interconnection tree.
- Recursion is shown if the interconnection tree contains a node which already exists on the path to the root of the tree. The lower part of the tree is not shown for the sake of simplicity. Continue to the node which is closer to the root.

The interconnection trees can only be opened up sequentially, not all at once with one click.

Displaying interconnection table

Select a TO in the interconnection table.

The associated interconnection table is displayed below the interconnection overview. The input and output interconnections of all interconnected TOs are listed.

3.7.2 Implicit interconnection when creating

Description

Interconnections are generated automatically when the following TOs are created:

- Synchronous axis; synchronous object is interconnected to the synchronous axis (following axis)
- Output cam; this is interconnected to the master value
- Cam track; this is interconnected to the master value
- Measuring input; this is interconnected to the master value

These interconnections are also displayed in the interconnection overview (Page 57).

3.7.3 Interconnection using general interconnection screen form in SCOUT (for experts)

SCOUT offers the **Interconnection** view for interconnection of various technology objects.

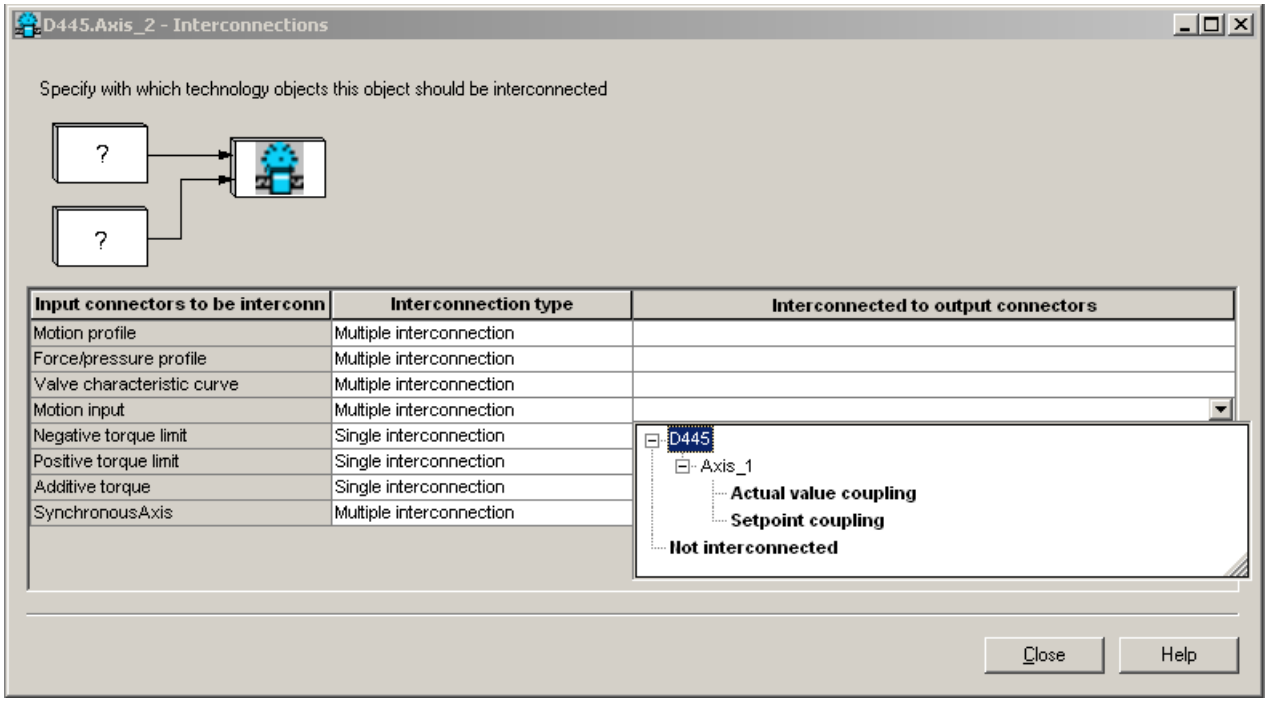


Figure 3-17 Interconnection in the SCOUT - axis example

On the left-hand side, all input-side interconnection interfaces of the selected object are listed, on the right-hand side, the output-side interconnection interfaces of the same type. Only the output-side interfaces displayed bold can be selected.

- When one of these lines is highlighted (clicked), you can open a tree that displays all interconnection possibilities.

Only those elements displayed **bold** can be selected.

The display of the devices and technology objects is used only for orientation purposes.

- Direct input can be made in the input line.

Only complete and correct inputs are accepted. The input field cannot be exited if incorrect inputs are made.

If the input-side interconnection interface has the characteristic of multiple interconnectability and can therefore be interconnected with several output-side interconnection interfaces, another line is added for the interconnection of the input-side interconnection interface after the interconnection with an output-side interconnection interface.

- An interconnection can be removed by deleting a line.

The value will be accepted when the input line is exited.

3.7.4 General properties of interconnection interfaces

Configuration of interconnection interfaces

The input-side interconnection interfaces are created:

- Implicitly with the instantiation of the TO
- Via technology-specific interconnection screen forms
- Via general interconnection screen forms

Enabling/disabling

The input-side interconnection interfaces are activated with a command on the associated TO or they are active by default.

In the case of multiple interconnectability of an input-side interconnection interface, enabling/disabling is carried out with a command for the technological function, e.g.:

- Selection of the cam in **_enableCamming()** via the **cam** parameter
- Selection of the technology object for the master value in the **_setMaster()** command via the **master** parameter
- Selection of the technology object for the reference value of the MotionIn interface in the **_run...basedMotionIn...()** command via the **mastertype.reference** parameter
- Selection of a profile or the cam in the **_run...Profile()** commands via the **profile** parameter

For information on this topic, please refer to the function manuals of the respective technology objects.

Indicator for status and interconnection values

The indicator for the status of the input-side interconnection and the interconnection values is defined on the TO for the individual interface types.

For information on this topic, please refer to the function manuals of the respective technology objects.

Validity

Interconnection interfaces have valid values when the interconnection is established and the interconnection values have the 'valid' status; invalid interconnections / interconnection values have the value 0.

If a (virtual) axis is set to simulation, the output-side setpoints will continue to be updated, the actual values will be frozen.

The status available in the system variables can be read out to determine whether the interconnection value or the substitute value is valid after ramp-up.

Alarm Response

In the case of a faulty interconnection, a technological alarm is only issued when a function on the interconnection interface is activated.

In the case of multiple interconnectability of an input-side interconnection interface, a reference TO must be selected.

Programmed functions will be aborted during the transition from **STOP U** to **STOP**; interconnection functions remain active during the transition from **STOP U** to **STOP**.

3.7.5 Interconnection interfaces on the TOs (for experts)

The input-side interconnection interfaces can only be interconnected with output-side interconnection interfaces of the same type.

TO	Interface	Type
Drive axis		
	Input-side interconnection interfaces:	
	Motion input	Motion
	Torque limit, positive	LREAL
	Torque limit, negative	LREAL
	Additive torque	LREAL
	Motion profile	MotionProfile
	Force/pressure profile	ForceProfile
	Valve characteristic (hydraulic axis)	ValveCharacteristics
	Output-side interconnection interfaces:	
	Motion setpoint	Motion
	Motion actual value with extrapolation	Motion
	Actual torque	LREAL
Position axis		
	Input-side interconnection interfaces:	
	As for drive axis	
	Output-side interconnection interfaces:	
	As for drive axis plus additionally:	
	Interface for output cam, cam track (implicitly interconnected with the axis by the system when creating an output cam, cam track)	Specific
	Interface for measuring input (implicitly interconnected with the axis by the system when creating a measuring input)	Specific
Following axis		
	Input-side interconnection interfaces:	
	As for position axis, plus:	
	Interface for synchronous object (implicitly interconnected by the system when creating a following axis/following object)	Specific
	Output-side interconnection interfaces:	
	As for position axis, plus:	
Path axis		
	Input-side interconnection interfaces	

TO	Interface	Type
	Interface as for the following axis, in addition:	
	Path motion	Specific
	Output-side interconnection interface	
	As for position axis	
Following object		
	Input-side interconnection interfaces:	
	Motion input	Motion
	Cam	CAM
	Output-side interconnection interfaces:	
	Interface for slave axis (implicitly interconnected by the system when creating a following axis/following object)	Specific
Path object		
	Input-side interconnection interfaces:	
	Velocity profile	MotionProfile
	Output-side interconnection interfaces:	
	Path motion (path axis 1)	Specific
	Path motion (path axis 2)	Specific
	Path motion (path axis 3)	Specific
	Path synchronous motion	Specific
	Motion output (path object.x)	Motion
	Motion output (path object.y)	Motion
	Motion output (path object.z)	Motion
External encoder:		
	Input-side interconnection interfaces:	
	None	
	Output-side interconnection interfaces:	
	Interface for output cam, cam track (implicitly interconnected by the system when creating an output cam, cam track)	Specific
	Interface for measuring input (implicitly interconnected by the system when creating a measuring input)	Specific
	Motion setpoint	Motion
	Motion actual value with extrapolation	Motion
Measuring input		
	Input-side interconnection interfaces:	
	Measuring input interface (implicitly interconnected with the axis, external encoder by the system when creating a measuring input)	Specific
	Input interface 'Listening measuring input' (as of V4.0) (via general interconnection screen form)	Specific
	Output-side interconnection interfaces:	
	Output interface for 'Listening measuring input' (as of V4.0)	Specific

TO	Interface	Type
Output cam, cam track		
	Input-side interconnection interfaces:	
	Output cam interface (implicitly interconnected with the axis, external encoder by the system when creating an output cam, cam track)	Specific
Cam		
	Input-side interconnection interfaces:	
	None	
	Output-side interconnection interfaces:	
	Cam	CAM
	Motion profile	MotionProfile
	Force/pressure profile	ForceProfile
	Valve characteristic (hydraulic axis)	ValveCharacteristics
Temperature controller		
	No interconnection interfaces	
Addition object		
	Input-side interconnection interfaces:	
	Motion output 1	Motion
	Motion input 2	Motion
	Motion input 3	Motion
	Motion input 4	Motion
	Output-side interconnection interfaces:	
	Motion output	Motion
Formula object		

TO	Interface		Type
	Input-side interconnection interfaces:		
		Motion input 1	Motion
		Motion input 2	Motion
		Motion input 3	Motion
		LREAL input 1	LREAL
		LREAL input 2	LREAL
		LREAL input 3	LREAL
		LREAL input 4	LREAL
		DINT input 1	DINT
		DINT input 2	DINT
		DINT input 3	DINT
		DINT input 4	DINT
	Output-side interconnection interfaces:		
		Motion output 1	Motion
		Motion output 2	Motion
		Motion output 3	Motion
		LREAL output 1	LREAL
		LREAL output 2	LREAL
		LREAL output 3	LREAL
		LREAL output 4	LREAL
		DINT output 1	DINT
		DINT output 2	DINT
		DINT output 3	DINT
		DINT output 4	DINT
Fixed gear			
	Input-side interconnection interfaces:		
		Motion input	Motion
	Output-side interconnection interfaces:		
		Motion output	Motion
Sensor			
	Input-side interconnection interfaces:		
		None	
	Output-side interconnection interfaces:		
		Sensor value	LREAL
		Sensor value derivation	LREAL
		Motion output	Motion
Controller object			

TO	Interface		Type
	Input-side interconnection interfaces:		
		Setpoint	LREAL
		Actual value	LREAL
		Feedforward control	LREAL
		Motion input setpoint	Motion
		Motion input actual value	Motion
	Output-side interconnection interfaces:		
		Output value on motion output	Motion
		Output value	LREAL

3.7.6 Interconnection interface type 'Motion' (for experts)

The interconnection interface type **Motion** contains the motion vector (s, v, a), internal information such as validity status and, if applicable, modulo information.

Motion basis

The motion vector can have a different motion basis: position or velocity.

The position component is zero for the "velocity" motion basis.

A drive axis provides only the motion vector with the velocity motion basis on the output side.

Commands / command parameters (TO axis) or the configuration (TO additionObject, TO fixedGear) are used to consider/set the motion basis.

The components of the motion vector are maintained consistent at the output side of the TO axis, TO fixedGear (the derivation of the position value is the velocity value, the derivation of the velocity value is the acceleration).

The components of the motion vector are not maintained consistent at the output side of the TO additionObject and TO formulaObject.

The velocity and the acceleration can be specifically defined in this item for the position.

'MotionInput' input-side interconnection interfaces of the 'Motion' type

The TO type specifies whether a 'MotionInput' input-side interconnection interface of the Motion type can be interconnected once or several times.

The following technology objects have, for example, an input-side interconnection interface 'MotionInput' of the 'Motion' type:

- Drive axes
- Positioning axes
- Following axes
- Fixed gear
- Addition object
- Formula object
- Following object (master value)

Output-side interconnection interfaces of the 'Motion' type

The following technology objects have, for example, an output-side interconnection interface of the 'Motion' type:

- Drive axes
- Positioning axes
- Following axes
- Path object
- Fixed gear
- Addition object
- Formula object
- External encoder

3.7.7 Interconnection interface type 'LREAL' (for experts)

There is an interconnection interface of the LREAL type available for scalar variables.

Input-side interconnection interface on the axis 'TorqueLimitPositive' and 'TorqueLimitNegative' for specifying the torque limit.

To specify the **B+/B-** torque limits, the 'TorqueLimitPositive' and 'TorqueLimitNegative' input-side interconnection interfaces of the axis can be interconnected with output-side interconnection interfaces of the LREAL type of other TOs.

Prerequisite is the activation of the process-related field on the TO axis.

The input-side interfaces 'TorqueLimitPositive' and 'TorqueLimitNegative' cannot be interconnected several times and they cannot be distributed.

Output-side interconnection interfaces of the LREAL type have, for example:

- TO axis with the output-side interconnection interface 'Torque'
- TO formula object with the output-side interconnection interface 'LRealOut'
- Sensor TO
- Controller Object TO

'Torque' output-side interconnection interface on the axis for providing the actual torque

There is a 'Torque' output-side interconnection interface available on the axis for issuing the actual torque to other TOs.

Prerequisite is the activation of the process-related field on the TO axis.

The actual torque supplied by the drive in the additive process-related field is issued.

It is issued in the unit set on the axis.

The output-side interconnection interface can be interconnected several times.

The output-side interconnection interface 'ActualTorque' cannot be distributed.

'AdditiveTorque' Input-side interconnection interface on the axis for specifying an additive torque

There is a 'AdditiveTorque' input-side interconnection interface of the LREAL type available on the axis for specifying an additive torque to the drive using the additive process-related field.

To specify an additive torque depending on other technology objects, the input-side interconnection interface 'AdditiveTorque' can be interconnected with output-side interconnection interfaces of the LREAL type of other technology objects on the axis.

The input-side interface 'AdditiveTorque' cannot be interconnected several times and it cannot be distributed.

Output-side interconnection interfaces of the LREAL type have, for example:

- TO axis with the output-side interconnection interface 'Torque'
- TO formula object with the output-side interconnection interface 'LRealOut'
- Sensor TO
- Controller Object TO

3.8 Technology object trace

3.8.1 Introduction

You can monitor commands at a technology object and track access to the system variables and configuration data.

In SIMOTION, technology objects can be influenced by configuration data, system variables, and commands during runtime. In order to represent the influences on a technology object during runtime through the user program, a TO trace / object trace, which records the last n actions at the technology object and arranges these in chronological order, is available as of V4.2.

Note

Simultaneous use of the **Trigger through program call** trigger condition for the TO trace and the device trace is not permitted.

If the TO trace and the device trace are active simultaneously, different trigger conditions must be used.

The recording can be read out from the technology object on request and represented in the ES.

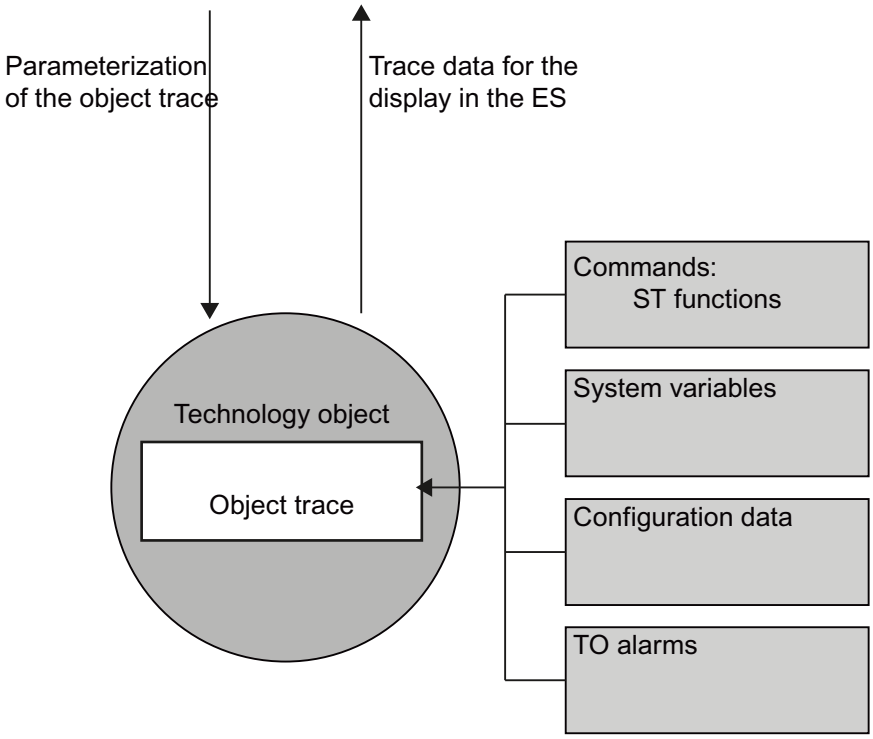


Figure 3-18 Principle of the object trace

The TO trace is used for the diagnostics of technology objects and displays the information and the events that influence a technology object during runtime in tabular form. The TO trace can be configured separately for each technology object.

It records commands and access to configuration data and system variables with their command and error statuses when the TO trace is read out. As of SIMOTION V4.4, alarms of the individual technology objects can also be recorded and an automatic trace is available. It can be used to trigger a recording automatically according to the trigger conditions immediately after a restart. You can parameterize the automatic trace in the Settings tab (Page 77) at **Save in the device (memory card)**.

The TO trace is represented in a table/list with a time stamp.

The TO trace is available for the following technology objects:

- Drive axis
- Position axis
- Following axis
- Path interpolation
- Following object
- External encoder
- Measuring input
- Output cam
- Cam track
- Addition object

- Formula object
- Sensor
- Temperature channel
- Fixed gear
- Controller object

3.8.2 Events

3.8.2.1 Events

The TO trace supports recording of events which influence a technology object during runtime.

Events

These events include:

- Command execution
- Writing of system variables and
- Writing or activation of configuration data
- Alarms

3.8.2.2 TO trace for PLCopen function blocks

Block calls of PLCopen function blocks are entered in the command buffer. Edge-triggered blocks are entered in the buffer with their rising edge. Level-triggered blocks are entered in the buffer each time the input level changes.

3.8.2.3 TO trace for commands or function blocks

You can use the TO trace to read out the last commands issued to the technology object.

As command execution advances, the more information can be read out concerning the relevant commands. This means you can determine the status (executed or aborted) of commands which have already been completed/terminated as well as any error code. As the TO trace takes a snapshot, this information is based on the situation at the time of reading out. The status is not updated automatically once the command is finished. You can trigger an update by performing a further readout.

Note

The command value on the TO displayed for commands or function blocks is the value transferred on the interface.

The value that is active on the TO, e.g. through limits, is not displayed.

The TO trace provides the following information:

Table 3-4 Information on commands in the TO trace:

Parameter	Description
Command type	The command name in plain text can be ascertained from the command type.
Time stamp 1	The RT time for when the command is issued is stored in the time stamp.
Time stamp 2	The RT time for when the command takes effect is stored in the time stamp.
Time stamp 3	The RT time for when the command is completed is stored in the time stamp.
Taskstack	The call point for the command in the user program can be ascertained from the taskstack.
Command status / block status	Status of the command. The phase of the current motion is also given for motion commands. Possible values are as follows: • BUFFERED • WAIT_SYNC_START • IN_EXECUTION • IN_ACCELERATION • IN_CONSTANT_MOTION • IN_DECELERATION • INTERPOLATION_DONE • EXECUTED • ABORTED
Error code	Return value when the command is finished. The error code is only valid if the motion status is EXECUTED or ABORTED.

Parameter	Description
n command parameters	The TO trace records the transfer parameters of the command. The programmed value and the command value on the TO are recorded for each command parameter.
n return values	All the return values of the commands are also recorded, in addition to the command parameters. Recording takes place in the user program when the command is finished.

3.8.2.4 TO trace for configuration data

The TO trace records write accesses

Table 3-5 Information on configuration data in the TO trace

Parameter	Description
Configuration data element	Name of configuration data element in plain text
Time stamp	The RT time for when the configuration data element is written is stored in the time stamp.
Target data range	The target data range shows the effectiveness of the configuration data element. Possible values are the NEXT data range or the ACTUAL data range.
Data type	Date type of configuration data element
New value	Value written in the configuration data element

3.8.2.5 TO trace for system variables

The write accesses to the system variables through the user program are recorded with the TO trace

Table 3-6 Information on system variables in the TO trace

Parameter	Description
System variable name	System variable name in plain text
Time stamp	The RT time for when the system variable is written is stored in the time stamp.
Data type	Data type of the system variables
New system variable value	Value written in the system variable

3.8.2.6 TO trace for alarms

The TO trace records alarms

Table 3-7 Information on alarms in the TO trace

Parameter	Description
Time stamp	The following times are saved in the time stamp: • Time alarm occurred • Time alarm was acknowledged
Alarm number	Number of the alarm

3.8.3 Call

The TO trace can be called in a number of ways

Call via the main menu

You can call the TO trace by selecting **Target system -> Technology object trace**. You need to select a device in the project navigator; otherwise, the menu command remains hidden. Where there is more than one device, you can call the TO trace individually for each device.

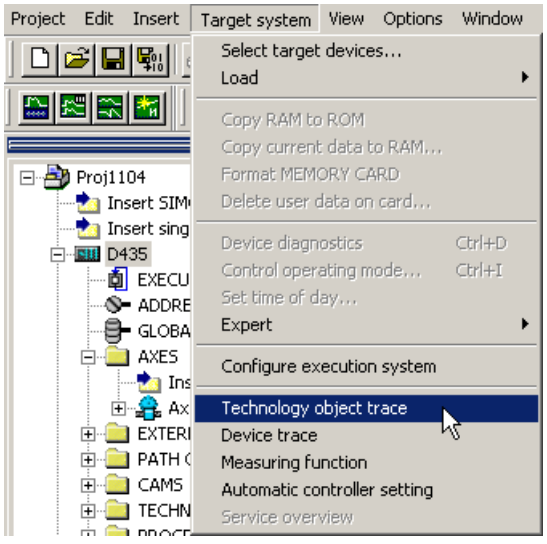


Figure 3-19 Call via the main menu

Call via the context menu

You can call the TO trace via the device's context menu.

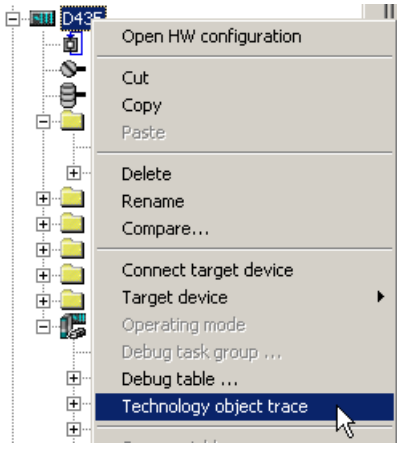


Figure 3-20 Call via the context menu

Call via the toolbar

The toolbar contains the button for calling the TO trace. The button is activated when a device is selected in the project navigator.

3.8.4 Parameterization

3.8.4.1 Overview of the parameterization

The parameterization and trace data are available both offline and online.

Only part of the overall functionality is available offline:

- Parameterization of the TO trace
- Displaying the latest TO trace to be uploaded
- Saving the data to a TO trace file
- Opening an existing TO trace

Additional online functionality:

- Parameterizing the TO trace and downloading it to RT
- Starting, stopping the trace recording
- Uploading and downloading the parameterization
- Delete the parameterization in the target system

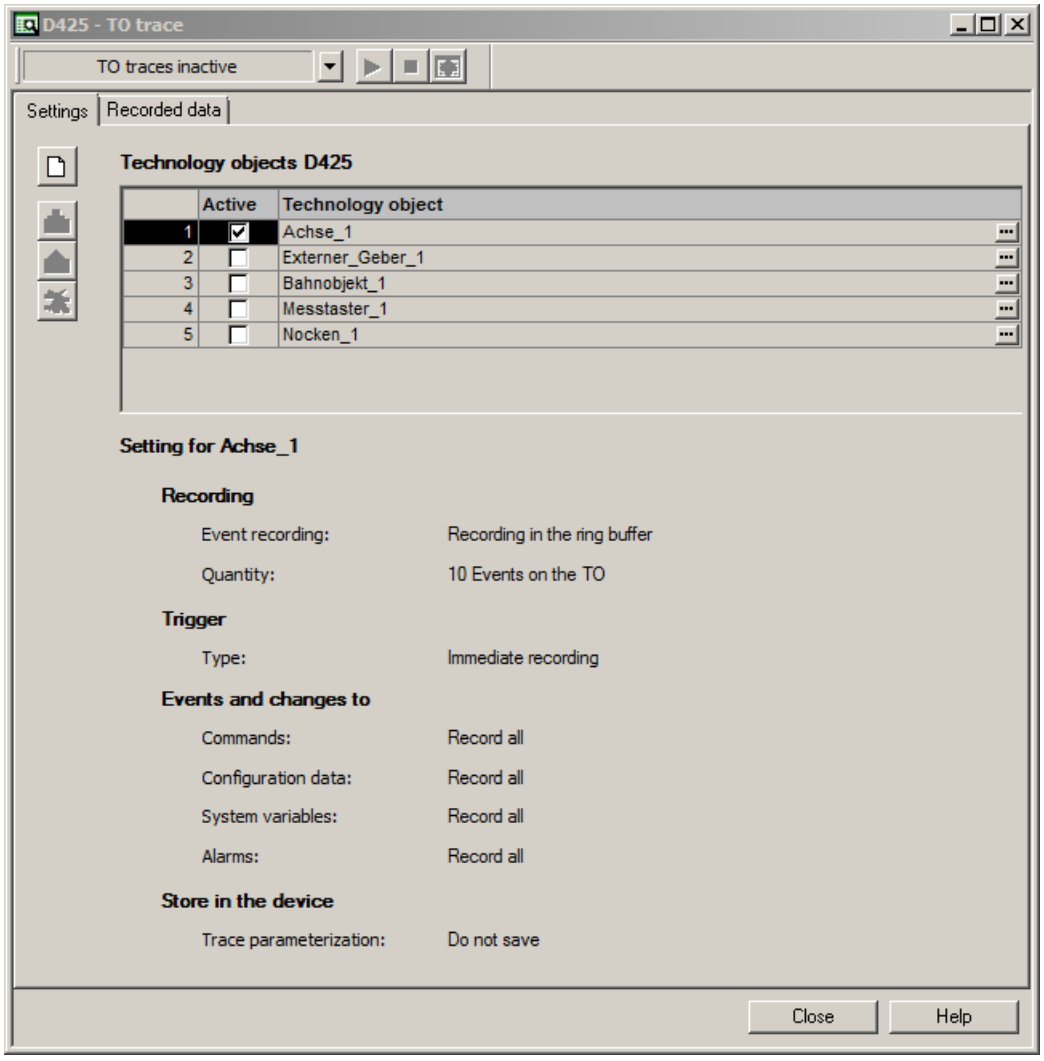


Figure 3-21 Dialog for parameterization

3.8.4.2 Toolbar

You control the recordings via the TO trace toolbar. The toolbar is displayed with the snap-in. This toolbar can be used to start or stop the trace for all technology objects and to upload a recording. The device to which the trace applies is fixed and cannot be changed. A separate trace window is opened for each device.

Table 3-8 Possible settings

Field/button	Description
TO traces inactive	Current state of the trace for all technology objects The field displays the state and availability of the TO trace. **Possible states:** • TO trace inactive (if all are complete or yet to start) • TO trace parameterization faulty (if at least one has an incorrect parameterization) • TO trace active (if at least one is running or waiting for a trigger)
	Opens a drop-down list to display the states of the individual TOs **Possible states:** • Inactive • Ready • Waiting for trigger • Running • Parameterization faulty • Recording finished
	Start TO trace
	Stop TO trace
	Upload data TO trace data can be uploaded at any time. If an endless trace is set at the time of the upload, the recording is appended to the existing trace in each case. If the time-limited trace is activated, the existing trace is overwritten.

3.8.4.3 Settings

The tab displays the technology objects created below the device. The settings of the Technology object trace event selection (Page 77) dialog box are displayed below the table for the technology object selected in the table.

Table 3-9 Possible settings

Function	Description
Menu commands	
	Reset parameterization
	Download parameterization
	Upload parameterization

Function	Description
	Delete the parameterization in the target system
Technology objects	All the technology objects created in the project are displayed under the **Technology objects** setting. To select a line, click any cell within it. The settings for a line are displayed under the technology objects. Use the  button to change the settings for a technology object. All the technology objects are deselected under the default setting. The relevant technology object must be selected for the recording.
No.	Number of the technology object
Active	Activates or deactivates the trace for the technology object
Technology object	Name of the technology object
	The Technology object trace event selection (Page 77) dialog box is called with this command.

3.8.4.4 Recorded data

The tab displays the recorded data and enables results to be saved and loaded in XML format.

Table 3-10 Possible settings

Function	Description
Menu commands	
	Load result
	Save results in XML format.
Reference time	Time specification to which the time stamp of the recorded events refer. The absolute time of the controller is used for the TO trace recording. Through the specification of the absolute time of the controller from the device or system trace recording as reference time, TO trace recordings can be chronologically assigned to the trace recordings. The absolute time of the controller in the device or system trace recording is displayed with the measuring cursor.
Time specifications relative to the reference time	Activation or deactivation of the reference time for the display
Columns	
Type	Type of recording
Time stamp	Time stamp of recording
Technology object	The reference object
Event	The event recorded
Current status	Status of recording
Detailed information for the selection	Detailed information is displayed in tabular form. Chronological sequence of events is structured as follows for time stamp, code position, event and current status: - Issued - Has become effective - Completed/aborted

Note

With PLCopen blocks, the return values are also current values and change while the block is running. In addition, the time stamp for **issued** corresponds to the **execute edge** on the block.

3.8.4.5 Technology object trace event selection

Commands, configuration data, and system variables are always selected as the default setting. Signals can be deselected on the individual tabs.

Make the settings for the recording for the selected TO in the **Technology Object Trace Event Selection** dialog. You can select or deselect certain commands, configuration data, system variables and alarms for the recording in separate tabs. Activate or deactivate the checkbox of a node to select or deselect the whole of the branch beneath.

The commands are sorted and grouped in the same way in which they are displayed in the command library.

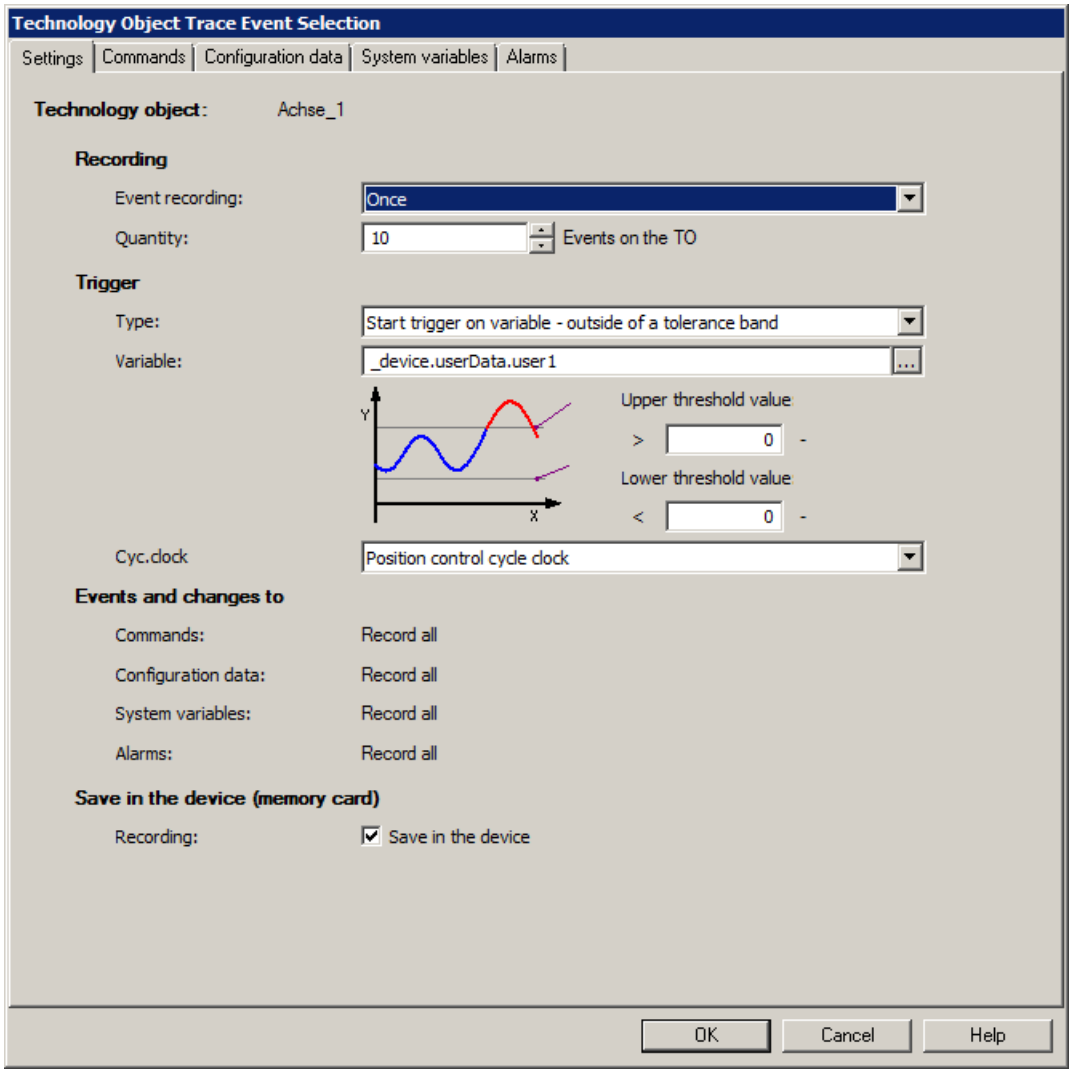


Figure 3-22 Technology Object Trace Event Selection

Table 3-11 You can set the following parameters:

Field/button	Meaning/instruction
Record	
Event recording	Recording of the events once or endlessly in a ring buffer With the setting **Once**, the trigger acts as the start trigger, with **Recording in the ring buffer** as the stop trigger.
Number	Buffer size of the number of events on the TO Up to 100 events can be recorded. The default setting is 10 events.
Trigger (the description is formulated for a start trigger and analogously is also valid for the stop trigger)	
Type	**Record immediately** Recording is started with the start of the TO trace.

<table>
<tr><th>Field/button</th><th colspan="3">Meaning/instruction</th></tr>
<tr><td></td><td colspan="3">Start trigger on variable ...
Recording is started when a certain trigger threshold of the variable value is reached. The threshold value to be reached and a specific system variable can be specified for the trigger. Select the cycle clock for this trigger type.
• Positive edge
If the entered trigger threshold is exceeded, the signal recording will be started. You can enter the trigger threshold under Threshold value.
• Negative edge
If the entered trigger threshold is undershot, the signal recording will be started. You can enter the trigger threshold under Threshold value.
• Within a tolerance band
The signal recording is started when the variable value enters a tolerance band. Enter the lower and upper threshold value.
• Outside a tolerance band
The signal recording is started when the variable value leaves a tolerance band. Enter the lower and upper threshold value.</td></tr>
<tr><td></td><td></td><td>Variable</td><td>Variable used as trigger</td></tr>
<tr><td></td><td></td><td>Threshold value</td><td>Value that is to be used as the trigger threshold</td></tr>
<tr><td></td><td></td><td>Upper/lower threshold value</td><td>Upper and lower threshold value for the tolerance band</td></tr>
<tr><td></td><td></td><td>Cycle clock</td><td>Cycle clock in which the monitoring is to be performed for when the trigger event occurs.
Servo cycle clock
The servo cycle clock is used as the basis for the trigger.
IPO cycle clock
The IPO cycle clock is used as the basis for the trigger.
IPO2 cycle clock
The IPO2 cycle clock is used as the basis for the trigger.</td></tr>
<tr><td></td><td colspan="3">Start trigger through 'TraceTrigger1' program call and Start trigger through 'TraceTrigger2' program call
The recording starts when the program call is triggered by the trigger event. In MCC, use the "Activate Trace" command and in ST, you can set the tracecontrol[0].tracetrigger system variable to the value "Start".
Note
Simultaneous use of the Trigger through program call trigger condition for the TO trace and the device trace is not permitted.
If the TO trace and the device trace are active simultaneously, different trigger conditions must be used.</td></tr>
<tr><td></td><td colspan="3">Start trigger at code position
The recording of variables is triggered by a defined code position in the program once the code position has been executed in the program sequence. You can parameterize the code position at Program source, Program line and Trigger in the task.</td></tr>
</table>

Field/button	Meaning/instruction		
		Program source	Name of the program from which the program variables are to be recorded. Note that only the name of the source is to be entered without a preceding controller name.
		Program line	Enter here the program line of the code position where the recording is to be started. Note that only lines containing executable code or program-local variable declarations will result in a recording. Empty program lines, lines that only contain comments or program lines with global declarations will result in an error message. How to calculate the program line: 1. Open the cross-reference list. 2. Select a line or cell in the list. If the **Calculate program line...** button is selected then the line can be used for the trigger. 3. Open the dialog to display the program line using the **Calculate program line...** button. 4. Adopt the line number displayed in the dialog as the **program line** for the recording. **Note** With ST programs the number displayed in the cross-reference list under **Line/Block** can be adopted as the **program line**. With MCC or LAD/FBD programs the **program line** must always be calculated using the **Calculate program line...** button. The program line calculated here does not match the number displayed under **Line/Block**.

Field/button	Meaning/instruction		
		After the n-th pass	How often the code position is to be executed until the measured value is acquired. The measured value acquisition is performed cyclically after each n-th pass.
		Trigger in the task	Task in which the program variables are to be recorded. If you have assigned the program to several tasks, only the task selected here will be considered for the recording.
Events and changes of			
	Commands	Displays whether all or only selected objects are to be recorded.	
	Configuration data	Displays whether all or only selected objects are to be recorded.	
	System variables	Displays whether all or only selected objects are to be recorded.	
	Alarms	Displays whether all or only selected objects are to be recorded.	
Save in the device (memory card) (as of SIMOTION V4.4)			
	Trace parameterization	Activation of saving in the device The trace parameterization is stored in the device or in the file system. Activate the TO trace in the Toolbar (Page 75) with the **Start TO trace** button. The TO trace is activated automatically during ramp-up. These measurements are retained after the controller is switched off. Read out the measurements with SIMOTION SCOUT (toolbar - **Upload data** button) or SIMOTION IT DIAG.	

3.8.5 Examples/applications

3.8.5.1 Saving the TO trace recording to the device (memory card)

The TO trace recordings can be saved retentively in the device with the **Save in the device** setting. In this way, a loaded TO trace can be automatically activated again the next time the controller is switched on.

Requirement

- The TO trace has been opened.
- An online connection has been established to the device.

Procedure

To parameterize a TO trace and save the recording in the device, proceed as follows:

1. In the Settings (Page 75) tab of the TO trace for a technology object, call the Technology Object Trace Event Selection (Page 77) dialog box.
2. Activate the **Save in the device** checkbox.
3. Close the Technology Object Trace Event Selection (Page 77) dialog box with **OK**.

4. Load the parameterization to the device.
   The icon for deleting the parameterization in the target system is activated.

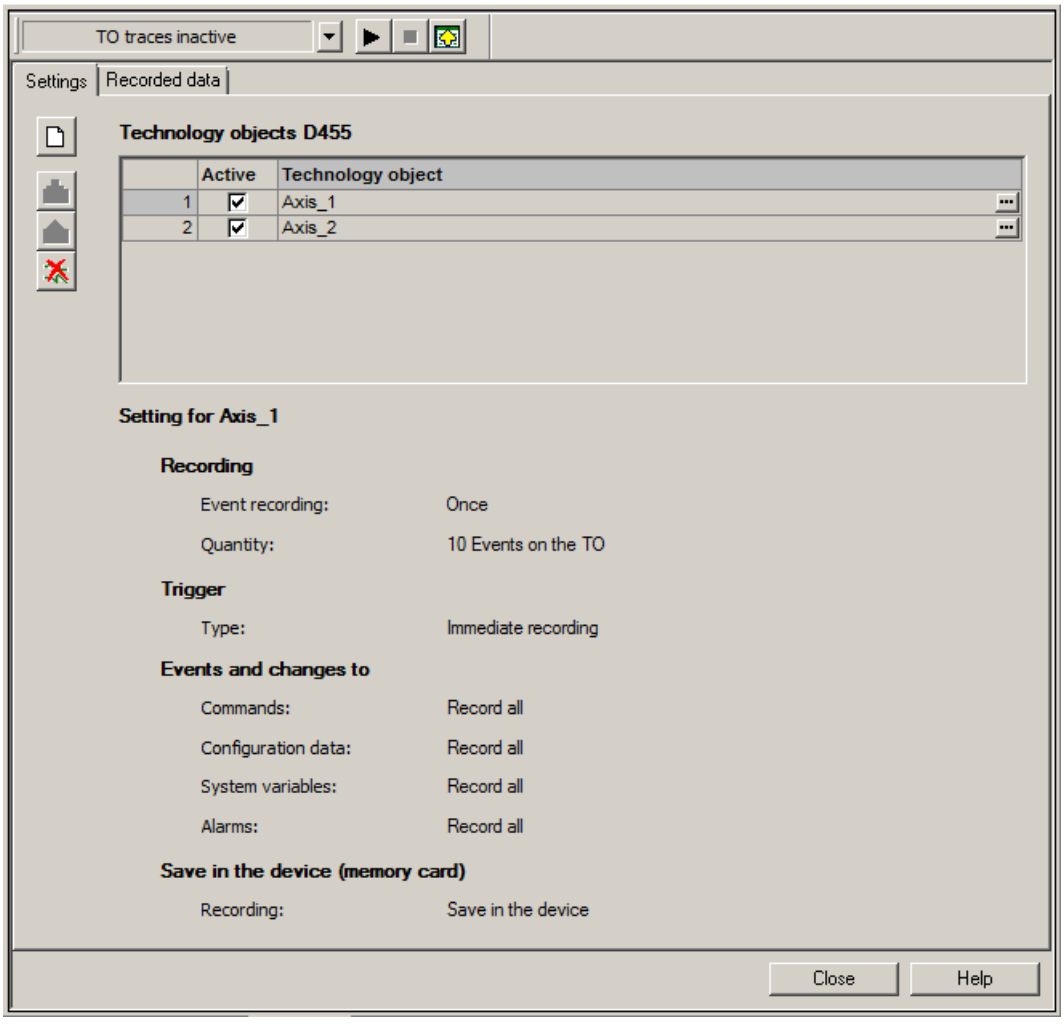


Figure 3-23 Settings tab for loaded parameterization in the device

5. Start the trace recording.
   The status is displayed in the toolbar.

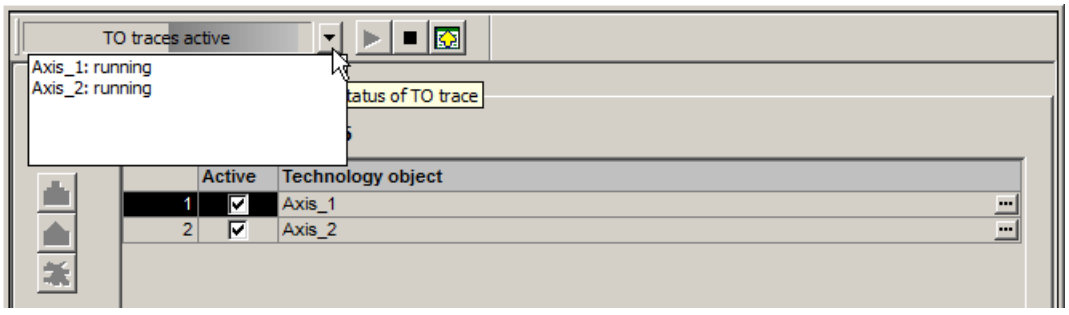


Figure 3-24 Status of the individual TOs

3.8.5.2 Reading out the TO trace recording from the device after a reconnection

TO trace recordings with the **Save in the device** setting can be read out with the TO trace even after disconnection of the online connection.

Note

A download of the TO trace parameterization may delete existing recordings in the device.

Requirement

The TO trace has been opened.

Procedure

To read out a recording saved in the device with TO trace, proceed as follows:

1. Establish an online connection to the device.
   The activated buttons to upload the parameterization and delete the parameterization in the target system indicate that a TO trace parameterization is available on the device.

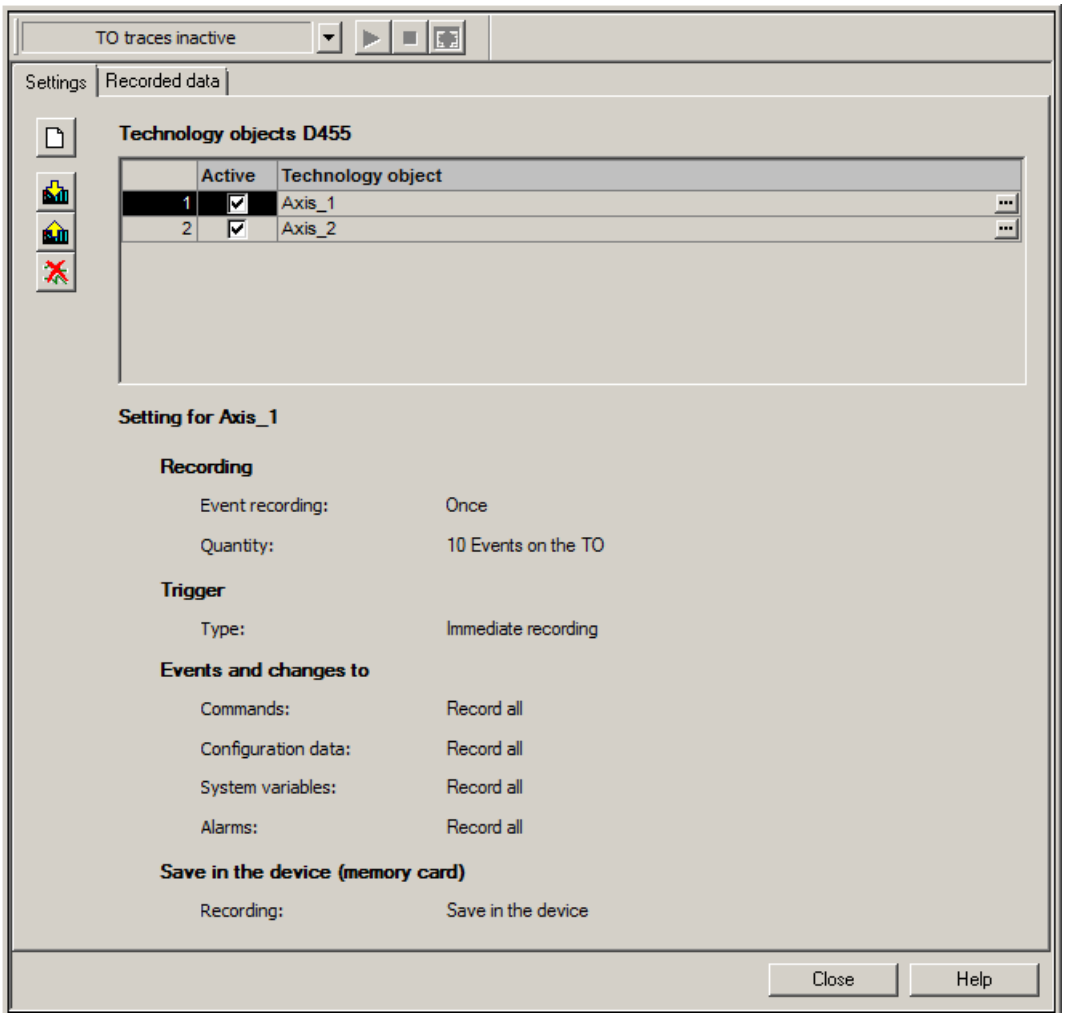


Figure 3-25 Settings tab after a reconnection when a TO trace parameterization is available on the device

2. Click the button to upload the parameterization.

Result

The elements of the offline parameterization are replaced by the online parameterization.

TO trace shows the status and, if required, the results gathered since the start of the trace. If the TO trace is still active in the device, further results are recorded.

See also

Technology object trace event selection (Page 77)

Call (Page 72)

Symbolic assignment (as of V4.2) 4

4.1 Symbolic assignment - introduction

Symbolic assignment of axis and drive

As of version V4.2, SIMOTION supports symbolic assignment to SINAMICS drive objects (DOs, drive objects) in the context of the configuration of technology objects (TOs) and I/Os.

This simplifies the configuration of the technological links including communication between the controller and the drive.

By means of symbolic assignment:

- Only compatible assignment partners are listed for selection in an assignment dialog
- The engineering system sets up communication between axis and drive automatically, along with the required PROFIdrive axis telegrams and the addresses used
- Telegrams are extended and assignments are created automatically in the drive depending on the selected TO technology (e.g. SINAMICS Safety Integrated)
- Axis and drive configuration can initially be carried out independently of one another
- Communication links are established automatically during the configuration of I/O variables on SINAMICS I/Os (telegrams and addresses are set up automatically and the I/Os are interconnected with the telegram).

Thus, other than symbolic assignment, no more configuration settings are required for communication. Since addresses no longer have to be configured, the connection is maintained even if addresses change.

Note

You can deactivate **automatic telegram configuration** and **automatic telegram adaptation** when configuring drive objects (DO drive, DO encoder, etc.) and in the dialog for telegram configuration.

If deactivated you will lose many of the benefits referred to above, we recommend it only in justifiable exceptional circumstances.

The TO axis, TO external encoder, TO output cam, TO cam track, and TO measuring input support symbolic assignment. It is also possible to establish symbolic assignment involving the onboard I/O of a SIMOTION D, a SINAMICS S110/S120 Control Unit, the TB30 and selected Terminal Modules.

The following components support symbolic assignment:

Module	Supports symbolic assignment
SIMOTION C, P, D	As of SIMOTION V4.2
Controller Extension • CX32-2 • CX32	As of SIMOTION V4.2
SINAMICS S110 CU305	As of SINAMICS V4.3
SINAMICS S120 • CU310 • CU310-2 • CU320 • CU320-2	 • As of SINAMICS V2.6.2 • As of SINAMICS V4.4 • As of SINAMICS V2.6.2 • As of SINAMICS V4.3
TB30, TM15 DI/DO, TM31	As of SIMOTION V4.2
TM41 (not DI/DO or AI)	As of SIMOTION V4.2
TM15, TM17 High Feature	As of SIMOTION V4.2

Note

The previous method of drive/axis and I/O configuration continues to be available. Symbolic assignment must be deactivated in such cases. Symbolic assignment is used by default for newly created projects.

Where projects are being upgraded to V4.2, symbolic assignment is deactivated by default and needs to be activated if required.

Symbolic assignment can be activated/deactivated in SIMOTION SCOUT via the menu "Project" > "Use symbolic assignment", see Using symbolic assignment (Page 116).

Adaptation

In addition to the benefits offered by symbolic assignment, configuration is also made easier as of SIMOTION V4.2 with the addition of automatic adaptation of SINAMICS S120 data. When SIMOTION devices ramp up, reference variables, along with drive and encoder data for SINAMICS S120, are transferred automatically for the configuration data of the SIMOTION technology objects "Axis" and "External Encoder". This data no longer has to be entered in SIMOTION.

Events, after which an adaptation is carried out:

- When the controller boots
- STOP->RUN transition, if no adaptation has been made yet
- In the case of a restart of a TO
- _activateTO
- For _enableAxisInterface, if the status "not adapted" is present at the interface

- Via a user command (driveControlConfig)
- After a data set changeover; encoder changeover does not cause adaptation

For further information, see Setting as a real axis with digital drive coupling in the Axis TO Electrical/Hydraulic, External Encoder Function Manual.

4.2 Symbolic assignment of TOs

4.2.1 Assigning the TO axis and TO external encoder

Description

In the axis wizard, the assignment dialog shows the available drives which can be symbolically assigned to the axis.

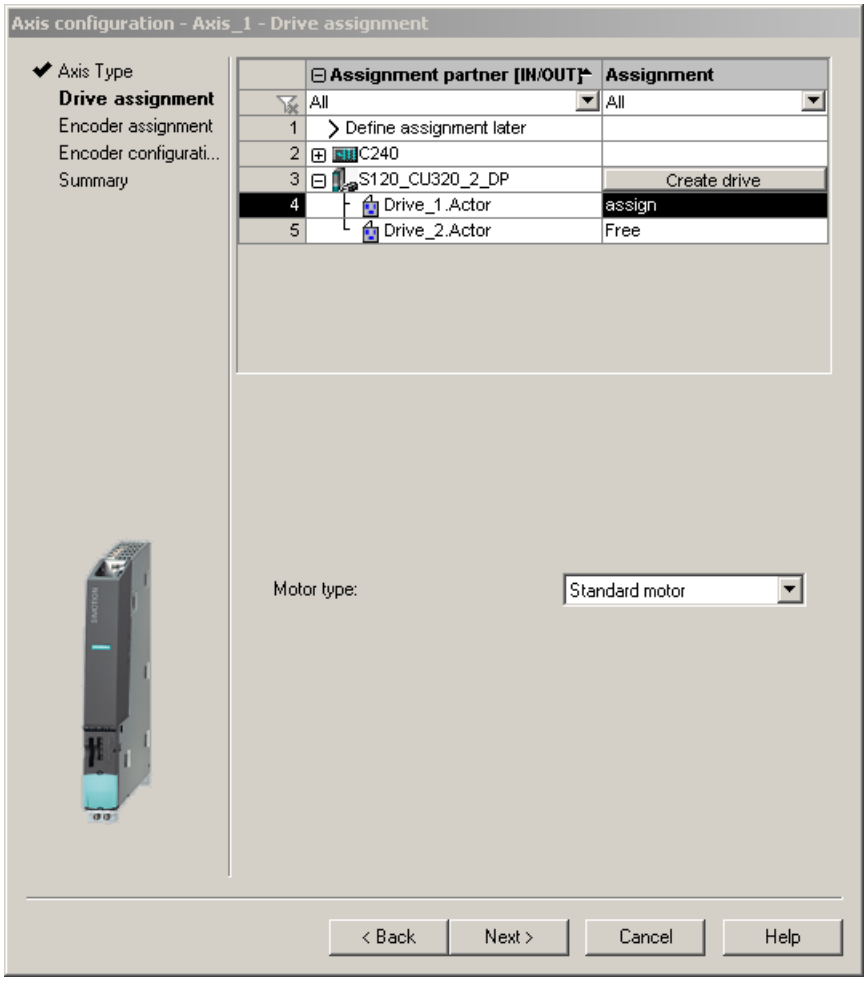


Figure 4-1 Assign axis and drive

Setting options

The following setting options are available:

- Assign drive
  Assign a previously configured drive
- Assign later
  The axis should not be assigned to a drive until a later point in time.
  Accordingly:
  - A programmer can use technology objects (Axis TO, for example) to make all the necessary configuration settings for the PLC and motion control functions and load them to the device; drive know-how is not required.
  - The drives can be configured and optimized separately by a drive expert and then, at a later point in time, the technology objects can be symbolically assigned to the drive objects via an interconnection dialog
- Create drive
  This function, which can be accessed in the assignment dialog, creates a new drive on an existing drive unit (e.g. S120 CU320-2 or SINAMICS Integrated) and assigns it to the axis directly. This allows the axis, including the drive, to be created in a single operation. It is not necessary to configure a drive before creating an axis.

Note

If you want to operate the drive directly via an application and not via a technology object, you must deactivate the "Automatic telegram setting" on the drive object, see also Message frame configuration (Page 110).

Encoder assignment

With a position axis, encoder 1 is also created at the Axis TO (motor encoder) and automatically assigned to the first encoder on the drive.

If encoder 2 (direct encoder) is created at the Axis TO, it is assigned to the second encoder of the drive control. See also axis wizard overview.

On completion of the axis wizard, the symbolic drive assignment can be viewed using the following methods:

- Via the "configuration" of the axis
- Via the address list (view all addresses)

The assignment dialog can be called again from these dialogs using the " ... " button.

Instead of calling the assignment dialog, it is also possible to edit the input field containing the symbolic name directly.

Technology data block (TDB) and drive safety data block (DSDB)

The activation of the technology data block and support for SINAMICS Safety Integrated functions by the technology object can be set in the configuration dialog of the Axis TO for functions under the "Change" button. The assignment is always made to the drive DO of the axis actuator. The system automatically generates a telegram extension and the BICO interconnection of the relevant SINAMICS parameters.

Assignment of the I/O signals on the Axis TO

For the assignment of I/O signals at the Axis TO, e.g. the inputs for the homing output cam or hardware limit switches, call the assignment dialog from the axis dialogs or from the address list (view all addresses) by clicking the " ... " button.

The detailed structure and the identifiers of the component types for symbolic assignment can be found in Assignment types for symbolic assignment (Page 696).

For further information about drive assignment, see the Axis Function Manual.

Address list

The address list in the "All addresses" view provides an overview of the assignments to all interfaces of the Axis TO. From this view, the assignments can also be changed via the assignment dialog (... button).

See also

Assigning I/O variables via the address list (Page 100)

Using the assignment dialog (Page 105)

4.2.2 Assigning the TO measuring input

Description

The assignment dialog displays the available encoders can be symbolically assigned to the external encoder.

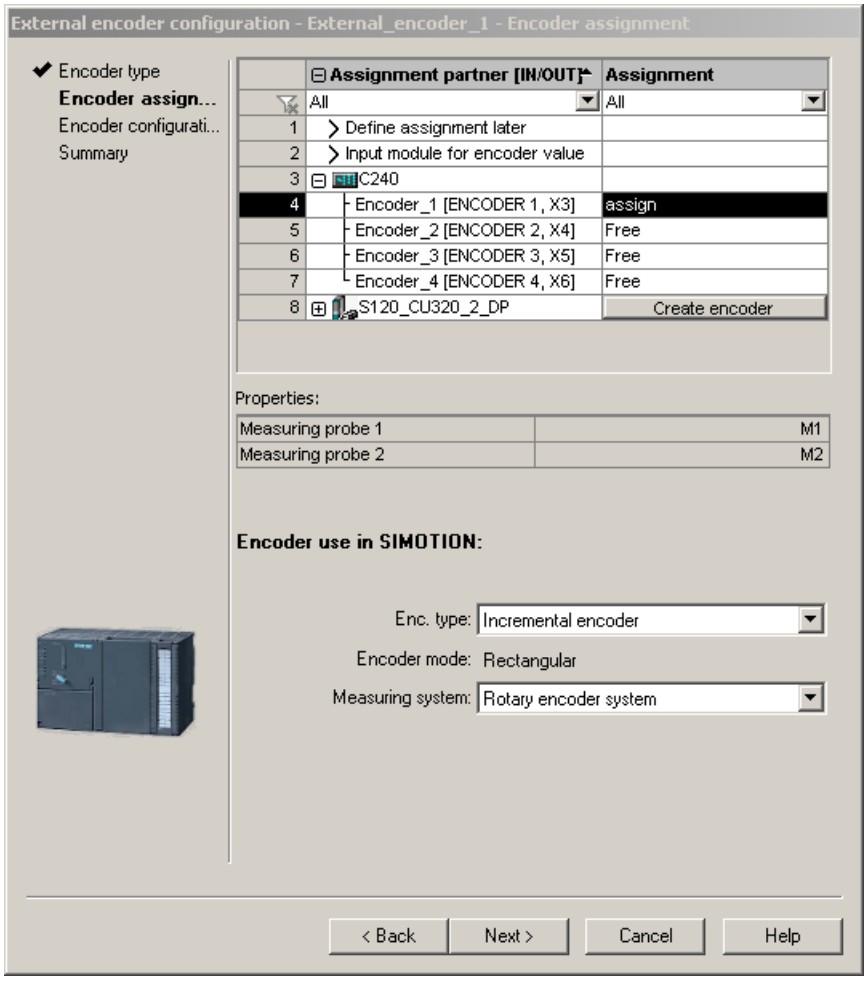


Figure 4-2 Assign External Encoder TO

For information about the axis wizard, which can also be used to configure the external encoder DO, see Overview of axis wizard.

See also

Using the assignment dialog (Page 105)

4.2.3 Assigning the TO output cam and cam track

Description

Before configuring the technology object, the functionality of the terminals must be configured accordingly (configuration of a DI/DO as a digital output, for example).

The configuration settings for the technology objects to access I/Os are made symbolically, whereby only "function-compatible" I/O channels are listed for selection in the assignment dialog.

Example 1:

Only MI (measuring input) type symbolic assignments are listed for selection for the Measuring Input TO.

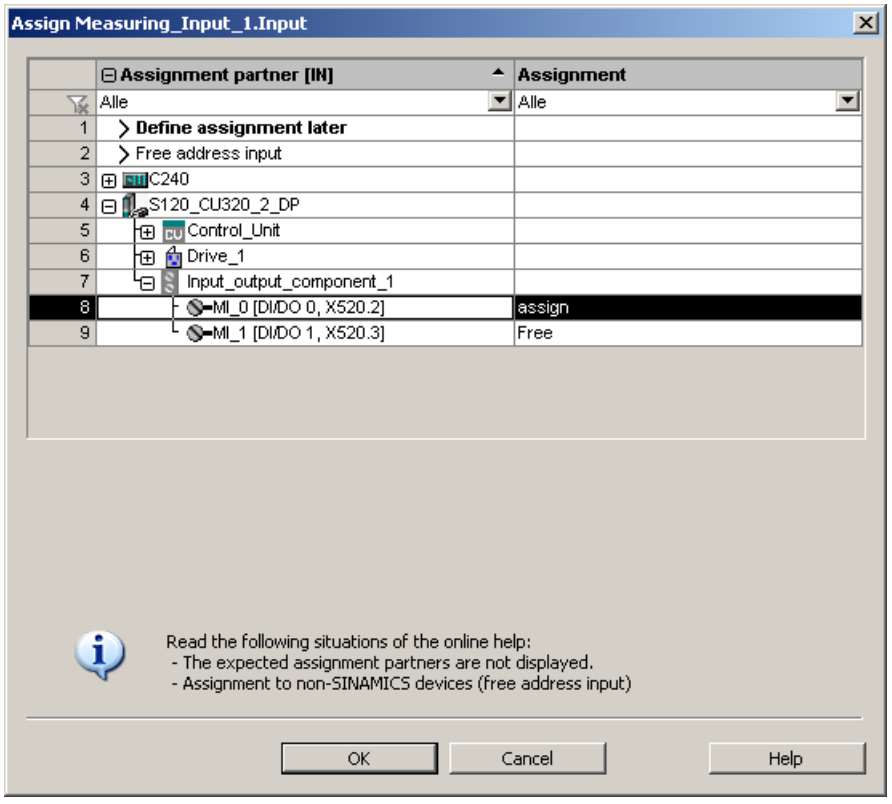


Figure 4-3 Assign measuring input

For further information, see the Output Cams and Measuring Inputs Function Manual:

- Assigning the TO output cam
- Assigning the TO cam track
- Assigning the TO measuring input

For information on the assignment dialog, see Using the assignment dialog (Page 105).

4.2.4 Assigning the TO sensor

Description

It might be necessary to configure the functionality of the terminals prior to assignment.

The analog inputs are listed for selection of the TO sensor in the assignment dialog.

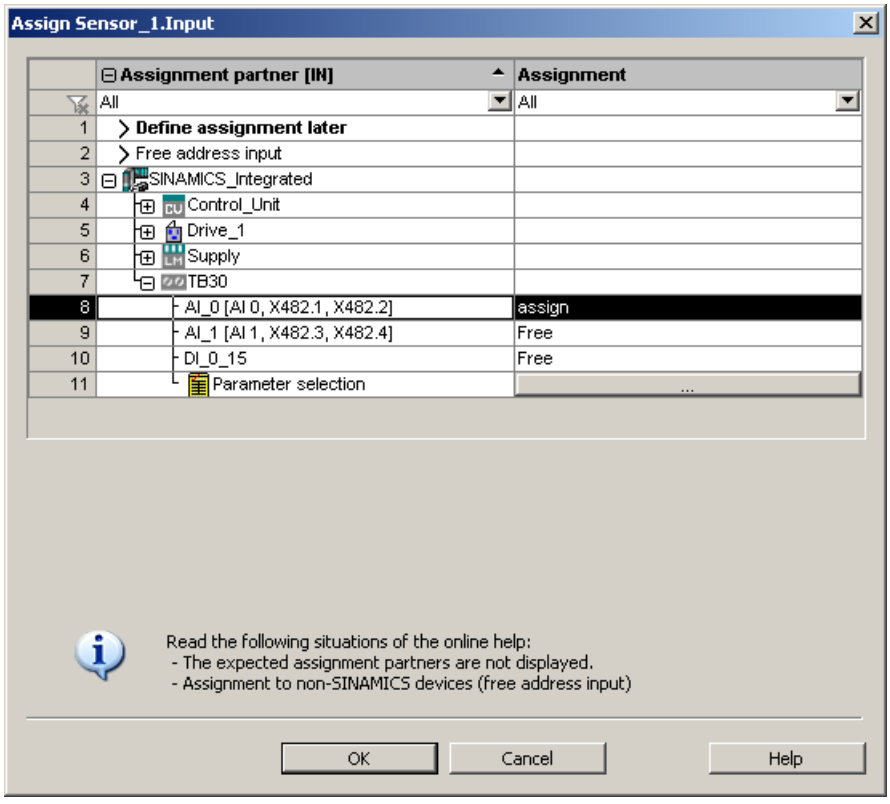


Figure 4-4 Assigning the TO sensor

For a description of the TO sensor, see the section titled TO Sensor in the "Additional technology objects" Function Manual.

The section titled Using the assignment dialog (Page 105) contains a description of the assignment dialog.

4.3 Symbolic assignment of I/O variables

4.3.1 Symbolic assignment of I/O variables to the PROFIdrive message frame of the Axis TO

Description

You can assign I/O variables from the address list which you require for display and diagnostic purposes, for example, to individual components (status word, for example) of the PROFIdrive telegram using the assignment dialog. Only components suitable for the data type of the I/O variable are displayed. If no data type is specified at the I/O variable, the data type this is determined by the assignment partner after selection.

To assign an I/O variable to an individual component of a PROFIdrive telegram

1. Open the address list and create a variable.
2. During I/O variable configuration, enter the identifier IN or OUT (or make a selection from the drop-down list box) under the I/O address for the direction.
3. Select the data type and open the assignment dialog by clicking the "..." button.
   The assignment dialog will then only show assignment targets which are suitable for this data type. If you do not select a data type, all parameters will be offered. The assignment dialog returns the parameter data type when it is closed.

4. Under Assignment partner, select the component (e.g. the word) that you wish to assign.
5. Click **OK** to assign the component.

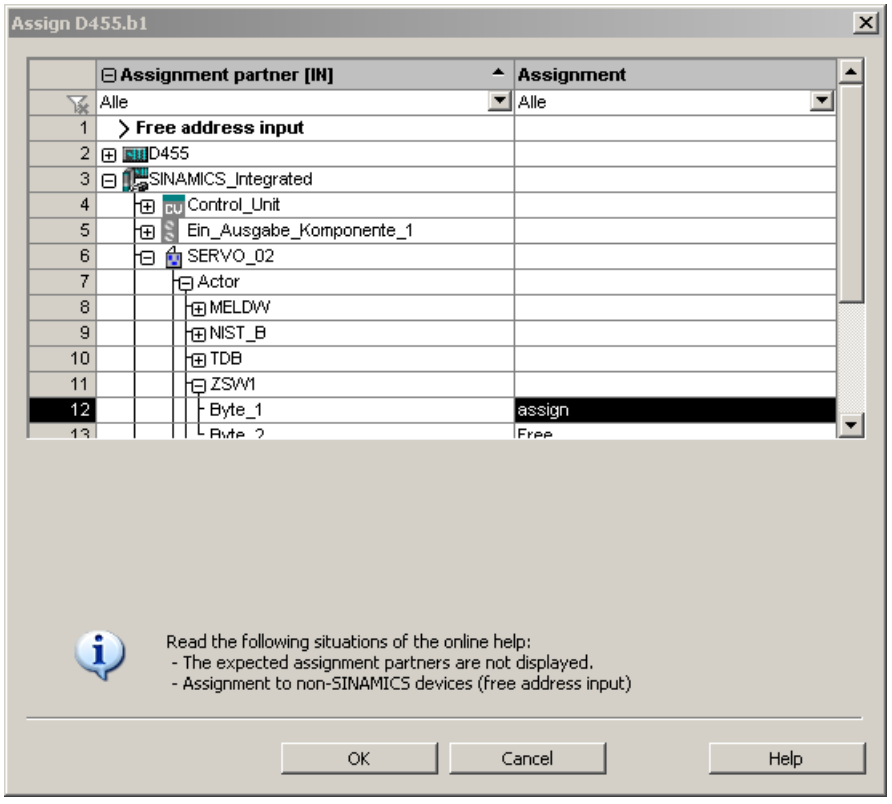


Figure 4-5 Assigning I/O variables to the PROFIdrive telegram of TO Axis

The detailed structure and the identifiers of the component types for symbolic assignment can be found in Assignment types in standard telegrams (Page 700).

4.3.2 Interconnecting parameters via BICO

Description

I/O variables from the address list can be assigned to drive parameters using the assignment dialog. Only parameters suitable for the data type of the I/O variable are displayed. If no data type is specified at the I/O variable, this is determined following parameter selection.

The system automatically extends the standard telegram for the drive for the transfer of the parameters.

BICO interconnection

You can interconnect a BICO parameter with a telegram by specifying the parameter numbers, the transmission width (WORD, DWORD), and the transmission direction in the assignment dialog.

As part of this, you can also select the parameter of a different signal source (SINAMICS DO). The parameter will then be transmitted automatically along with the telegram of the assignment partner.

How to assign I/O variables to drive parameters

1. Open the address list.
2. During I/O variable configuration, enter the identifier IN or OUT (or make a selection from the drop down list box) under the I/O address for the direction.
3. Select the data type and open the assignment dialog by clicking the "..." button.
   The assignment dialog will then only show assignment destinations which are suitable for this data type. If you do not select a data type, all parameters will be offered. The assignment dialog returns the parameter data type when it is closed.

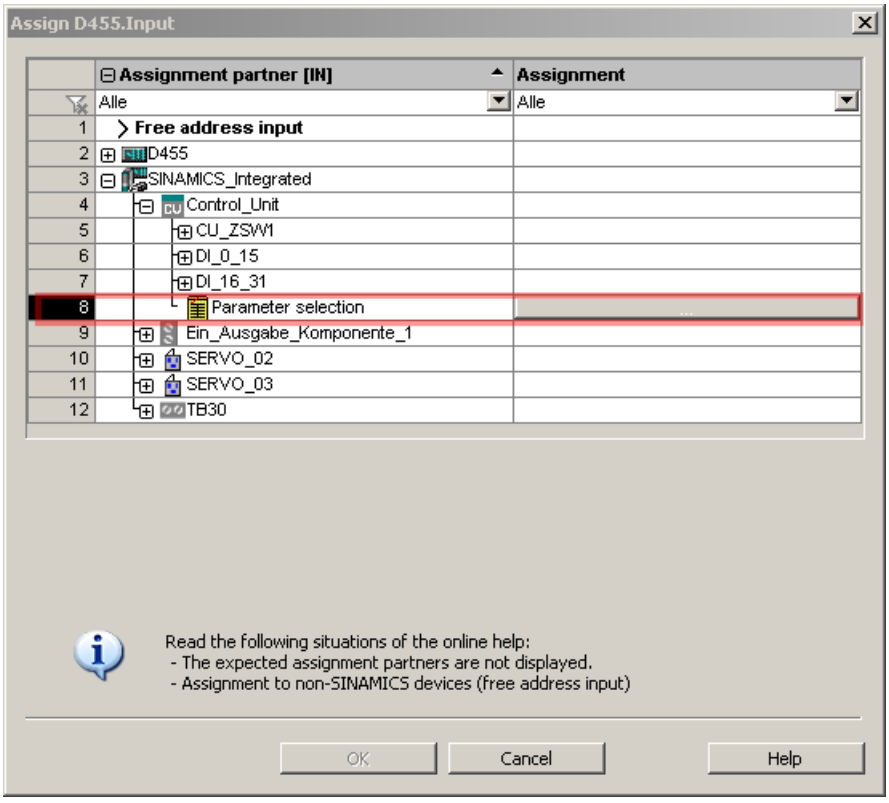


Figure 4-6 Assignment dialog for drive parameters

4. Click the ... button in the parameter selection line to open the parameter list.

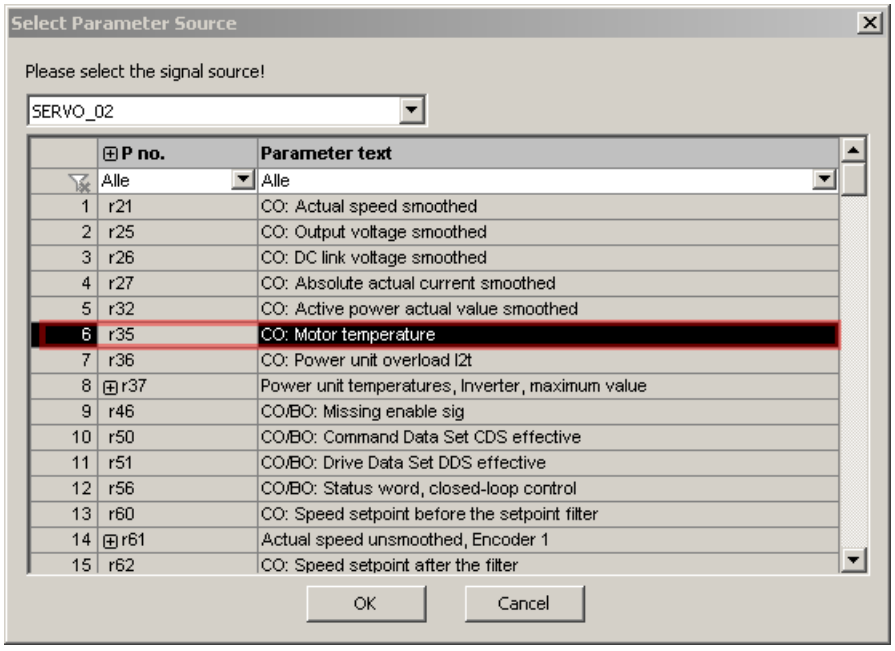


Figure 4-7 Dialog for parameter and DO selection

5. If necessary, select the signal source (SINAMICS DO) which makes the parameter available, followed by the parameter itself.
6. Click **OK** to accept the selection.

7. A SINAMICS parameter is assigned to the I/O variable in the assignment dialog.

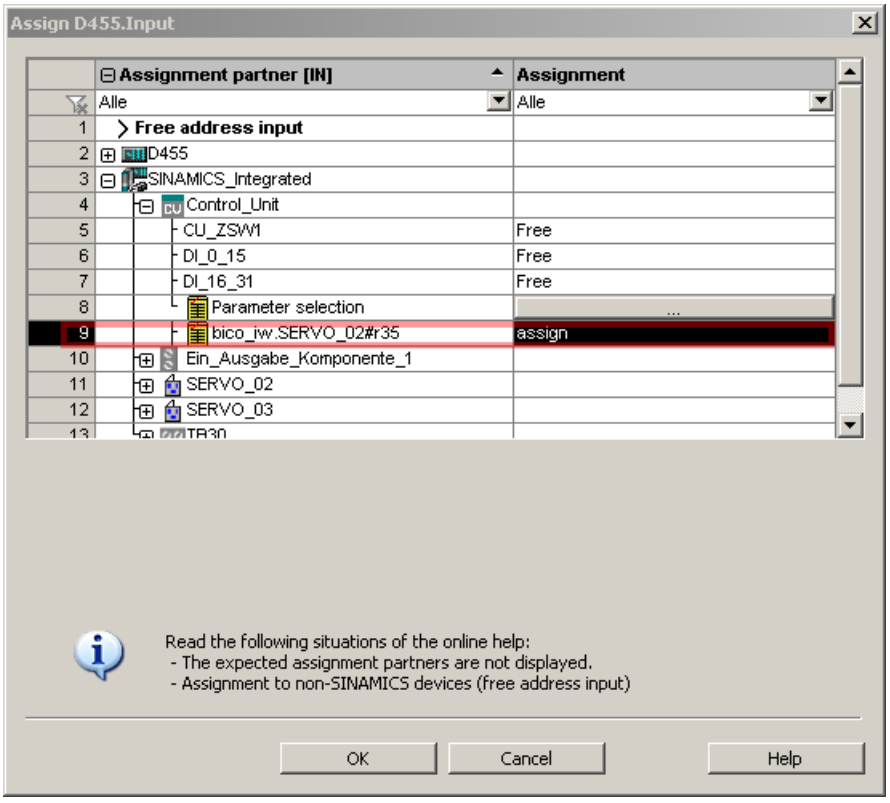


Figure 4-8 Assigned drive parameter

8. Click **OK** to accept the assignment.

Assignment types

The following table shows the possible types of assignment:

Name of the BICO interface	Data type	Direc-tion	Transferrable BICO parameters
BICO_IW.<parameter number>	WORD	Input	All CO parameters (BICO source)
BICO_QW.<parameter number>	WORD	Output	All CI parameters (BICO sink)
BICO_ID.<parameter number>	DWORD	Input	All CO parameters (BICO source)
BICO_QD.<parameter number>	DWORD	Output	All CI parameters (BICO sink)

Syntax of the assignment names:

- A number of parameters (separated by periods) are specified for outputs (SINAMICS side = received data) which can be interconnected with a number of BICO sinks.
- If the transferred parameter is on another DO, the DO name precedes the parameter. "#" is used as a separator between the DO name and the parameter.
- Individually transferred bits of a parameter appear in brackets [x].

4.3.3 Assigning I/O variables via the address list

Description

Prior to assignment, the functionality of the terminals might have to be configured accordingly (configuration of a DI/DO as a digital output, for example).

There are several ways to assign an I/O variable to terminals:

- Assignment of a terminal signal via the SIMOTION preferential interconnection. To do this, you must use the SIMOTION preferential interconnection for the corresponding input/ outputs of the SINAMICS DOs. The BICO interconnection is automatically executed.
- Assignment of a terminal signal via PZD interfaces, e.g. DI_0_15 or DO_0_15. The interfaces are, for example, assigned to PZD 2 (A_DIGITAL or E_DIGITAL).
  Please note for these signals that the BICO interconnection is not executed (e.g. for TB30, TM31 or TM15DIDO), although a q of the appropriate length is generated.

To activate the SIMOTION preferential interconnection for all terminals in a SINAMICS DO

1. In the project navigator, open the folder inputs/outputs of the relevant drive DOs.

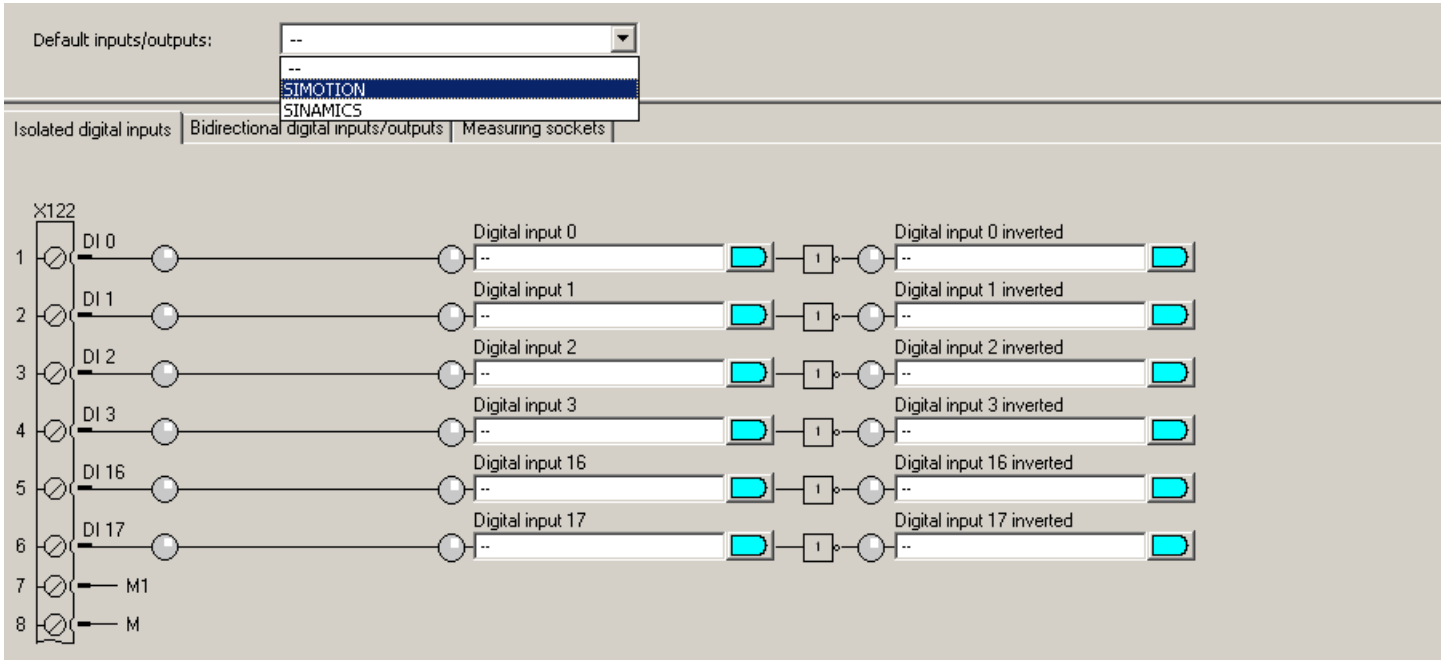


Figure 4-9 SIMOTION preferential interconnection

2. Select the entry **SIMOTION** in the **Default inputs/outputs** drop-down list box. All inputs/outputs are assigned to the SIMOTION preferential interconnections.

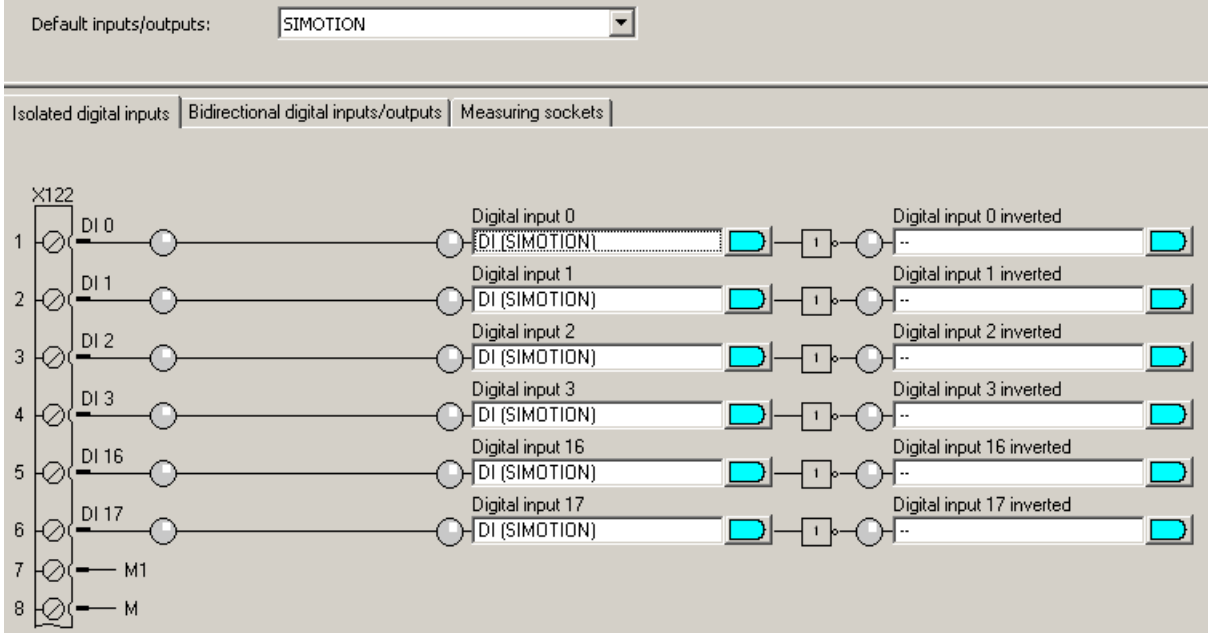


Figure 4-10 SIMOTION preferential interconnection active

With this setting, the BICO interconnections will be performed automatically.

If you wish to undo the interconnection, you must select SINAMICS in the drop-down list box for pre-assignment of the inputs/outputs.

To switch a single terminal

1. In the project navigator, open the folder inputs/outputs of the relevant drive DOs.
2. Click the tab in the foreground which shows the required inputs/outputs.
3. Click in the BICO field to display the possible interconnection targets.

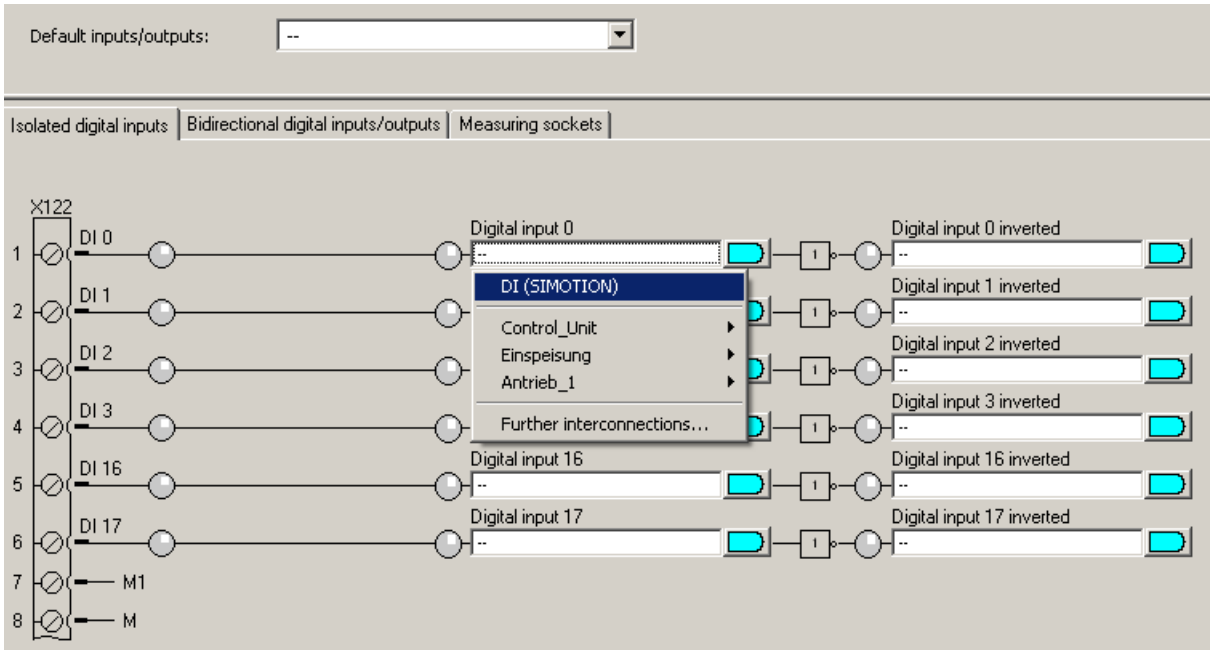


Figure 4-11 Setting the SIMOTION preferential interconnection locally

4. Select, for example, the entry DI (SIMOTION), to assign a digital input.

This is the way to proceed with a symbolic assignment with the SIMOTION preferential interconnection:

1. Activate in the drive DO the SIMOTION preferential interconnection (see above).
2. Open the address list in SCOUT.
3. For the configuration of the I/O variables, enter the identifier IN or OUT (or make a selection from the drop-down list box) under the I/O address for the direction, select the data type, and open the assignment dialog by clicking the " ... " button.

4. The assignment dialog will then only show assignment targets which are suitable for this data type.
   If no data type is selected, every known assignment parameter is listed for selection (e.g. bit, WORD, DWORD, input/output terminals). The assignment dialog returns the data type of the assignment partner when it is closed.
5. Select the appropriate terminal signal and click **OK**.

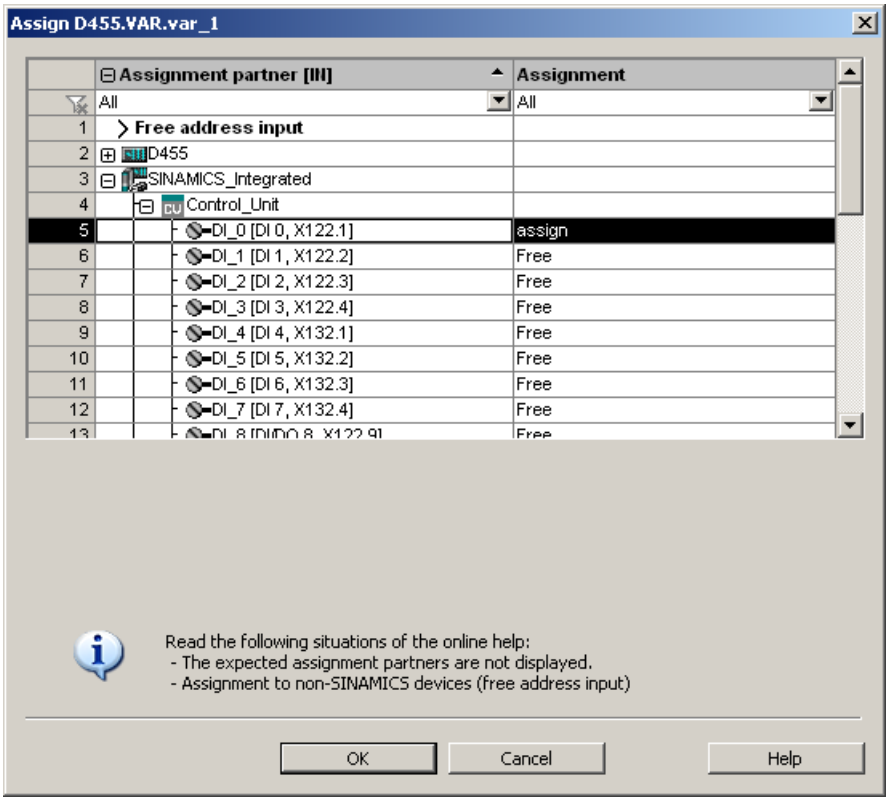


Figure 4-12 Assigning an I/O variable

Note

If the SIMOTION preferential interconnection is not activated, you only see entries like DI_0_15 in the dialog (see below). In this case, only interconnect the I/O variable to one bit in DI_0_15. The BICO interconnections must then be created manually, otherwise the interconnection does not function.

This is the way to proceed with a symbolic assignment of telegrams via PZDs:

1. Open the address list in SCOUT.
2. For the configuration of the I/O variables, enter the identifier IN or OUT (or make a selection from the drop-down list box) under the I/O address for the direction, select the data type, and open the assignment dialog by clicking the " ... " button.

3. Select the interface, to which you wish to assign an I/O variable, e.g. DI_0_15.
   You can also selectively assign the variable to a bit.

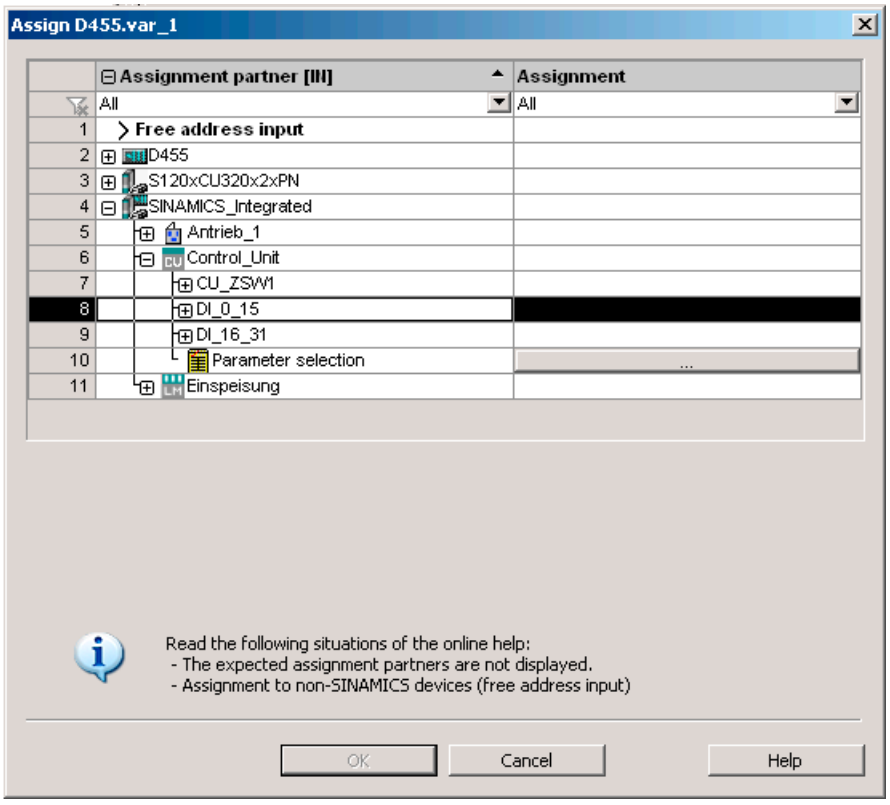


Figure 4-13 Assignment dialog for PZDs

4. Click **OK** to accept the assignment.
5. Now change to the dialog **communication** of the relevant SINAMICS DO. You can now see the individual bits of the PZD (e.g. E_Digital or A_Digital) listed there.
6. Interconnect the corresponding bit of the PZD with a signal, e.g. **External warning active**. This will execute the BICO interconnection.

Note

When you have activated the symbolic assignment, the telegrams 39x will be automatically assigned for the SINAMICS Control Units (at least telegram 390). The following interconnection of the terminals is, however, not executed in line with the original definition of the telegrams (without symbolic interconnection) since the terminal configuration is defined by it with the symbolic interconnection and managed by the symbolic interconnection.

Assignment of I/O variables with substitute values

Substitute values cannot be specified for I/O variables of data type BOOL. If you do need substitute values, however, proceed as follows:

1. Assign a Bool-type variable to, for example, the actuator interface of a drive, Drive_1.Control_Unit.DI_0_15
2. Create a global variable (with WORD as the minimum data type).
3. Assign the actuator interface of the drive to the variable in the assignment dialog, Drive_1.Control_Unit.MELDW.
   Bit1 of the substitute value must then contain the substitute value for the Bool variable.
   You can assign a substitute value to a BICO parameter using the same kind of method.

Additional, higher-level interfaces for assigning substitute values are available for various SINAMICS DOs, see SINAMICS assignment types (Page 698).

See also

Assigning the TO axis and TO external encoder (Page 89)

Assigning the drive I/O symbolically (Page 308)

4.4 Working with the assignment dialog

4.4.1 Using the assignment dialog

Description

A SIMOTION object can be assigned to a SINAMICS object using the assignment dialog.

You can call the assignment dialog from various technology object configuration dialogs and from the address list. The display of the dialog is dependent on the assignment for which you call it; for example, if an input, an output, or an actuator/encoder is to be assigned.

The analog outputs of a C240 cannot be interconnected symbolically. These must be interconnected using the input of addresses.

Signal or assignment type not found

The assignment dialog only shows assignment types which are suitable for the SIMOTION type. If you were expecting a different assignment type, check:

- The data type of the assignment type (e.g. for I/O variables)
- The configuration of the SINAMICS DO (input/output)
- Whether symbolic assignment is switched on (**Project > Symbolic assignment**)

Structure of the assignment dialog with symbolic assignment activated

The assignment dialog with symbolic assignment activated is available in the axis wizard, the address list, and TO configuration screen forms.

The assignment dialog has the following structure:

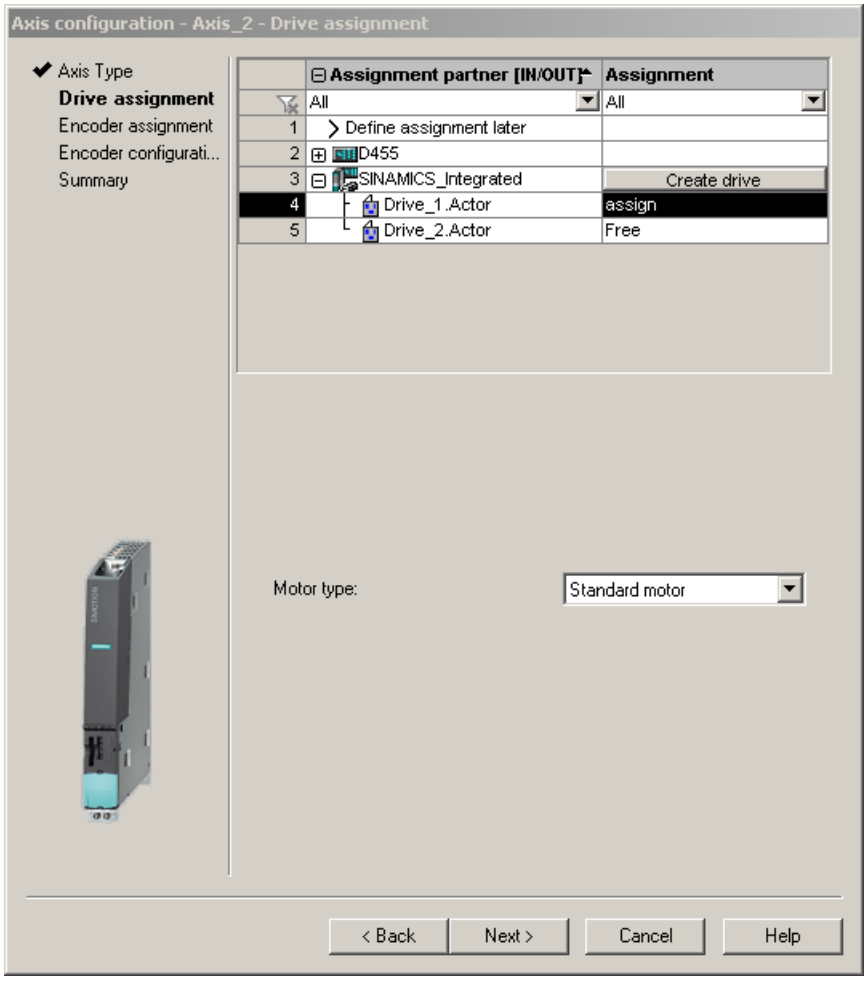


Figure 4-14 Assignment dialog with symbolic assignment activated

Field	Description
Title bar	The title bar contains the name of the SIMOTION interface or I/O variable to be interconnected. The dialog only shows the assignment parameters which are suitable for this interface.
Filter line	The filter line enables you to filter what is displayed in the table according to the criteria **all**, **free**, and **assigned**.
Assignment partner	This column lists all the devices and DOs. Click the "+" sign to expand the groups and display suitable interfaces, e.g. under the control unit. Note: If you assign an I/O variable to an I/O terminal of SINAMICS modules, you must still execute the BICO interconnection in the drive DO when using the interfaces DI_0_15 or DO_0_15 (name can vary depending on the number of terminals), see also Assigning I/O variables via the address list (Page 100).

Field		Description
	Define assignment later	Select **Assign later** if you want to assign the SIMOTION interface at a later time.
	Free address input	Select this entry if you need/want to enter the address for the assignment partner yourself (e.g. for devices not assigned symbolically, hydraulic axes, or analog modules). You then have to read out the address in HW Config and enter it here.
	Parameter selection	Under Parameter selection you can select parameters of the SINAMICS DO. See also Interconnecting parameters via BICO (Page 96).
Assignment		This column shows whether the SIMOTION interface is already assigned: • Blank or empty, the interface is still unassigned. • Assign; the interface is assigned to the selected assignment partner. • <assigned interface>; the interface is already assigned. The assigned SIMOTION interface is shown.
Create drive (Axis wizard - Assign drive)		Click the button to call the drive wizard for creating and configuring a SINAMICS drive.
Create encoder (Axis wizard - Assign encoder)		Click on the button to call the wizard for configuring the encoder DO.

Structure of the assignment dialog with symbolic assignment deactivated

The assignment dialog with symbolic assignment deactivated is only available in the axis wizard (drive assignment and encoder assignment).

The assignment dialog has the following structure:

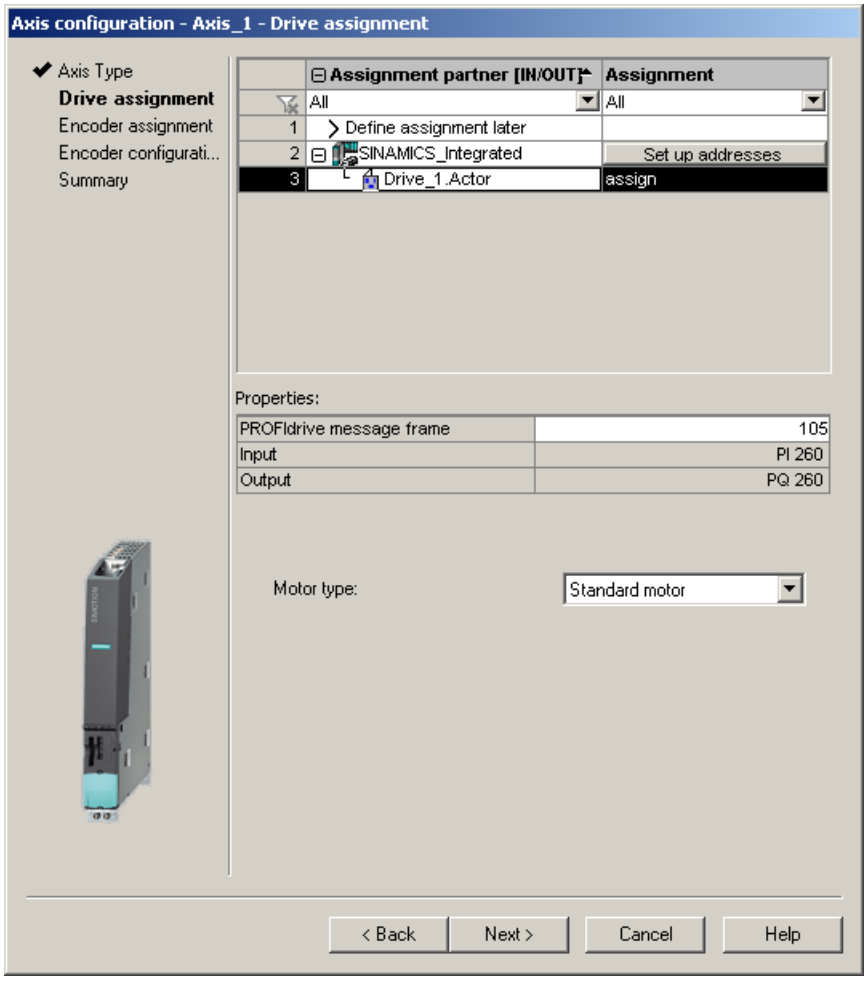


Figure 4-15 Assignment dialog

Field		Description
Title bar		The title bar contains the name of the SIMOTION interface or I/O variable to be interconnected. The dialog only shows the assignment parameters which are suitable for this interface.
Filter line		The filter line enables you to filter what is displayed in the table according to the criteria **all**, **free**, and **assigned**.
Assignment partner		This column lists all the devices and DOs. Click the "+" sign to expand the groups and display suitable interfaces, e.g. under the control unit.
	Free address input	Select this entry if you need/want to enter the address for the assignment partner yourself (e.g. for devices not assigned symbolically or hydraulic axes). You then have to read out the address in HW Config and enter it here.
	Parameter selection	Under Parameter selection you can select parameters of the SINAMICS DO.
Assignment		This column shows whether the SIMOTION interface is already assigned: • Blank or empty, the interface is still unassigned. • Assign; the interface is assigned to the selected assignment partner. • <assigned interface>; the interface is already assigned. The assigned SIMOTION interface is shown.

Field		Description
	Set up address	This button is for transferring the addresses, message frames, etc. of devices that are not symbolically assigned to HW Config.
Properties		In this table you can enter settings for the drive or encoder.

Assignment dialog with symbolic assignment deactivated or for drives which do not support symbolic assignment

The drives are only fully visible once you have performed "Set up addresses" (using HW Config). This affects SIMODRIVE or MASTERDRIVES drives, for example, and the PROFIBUS modules ADI4 or IM174. See also AUTOHOTSPOT.

Free address input

In addition to the symbolic assignment, you can also enter a hardware address directly under "Free address input". This is necessary, for example, if the hardware components do not support symbolic assignment.

The address entry in the text box is checked and depends on the data type

specified when calling the dialog.

- If a data type was not entered when the dialog was called, the correct syntax of the address is checked and the data type is determined on the basis of the address. It is then entered in the address list, for example.
- The address of an output variable cannot be entered for input variables, and vice versa.

4.5 Setting up addresses and message frames

4.5.1 Setting up addresses and message frames - overview

Overview

After all SINAMICS components have been configured, addresses must be determined for the process data exchange between the drive and the controller.

This procedure depends on whether **symbolic assignments** are used.

- With **symbolic assignment**, the addresses are determined automatically by the engineering system; see the section titled Setting up communication for symbolic assignment (Page 110).
- Without **symbolic assignment**, the addresses must be determined manually; see the section titled Message frame configuration (Page 110).

See also

Acyclic communication with the drive (Page 512)

4.5.2 Setting up communication for symbolic assignment

You can set up communication for symbolic assignment using the following methods:

- Via the SCOUT menu (call the following in the menu: **Project > Set up communication for symbolic assignment**)
- By downloading to the target system
- By saving the project and compiling changes

When you set up communication, the message frames, BICO interconnections, and addresses are set up for the entire project.

See also

Setting up addresses and message frames - overview (Page 109)

Message frame configuration (Page 110)

Downgrading to versions <V4.2 (Page 119)

4.5.3 Message frame configuration

Requirement

You have configured the drive unit.

Note

You only need to carry out these steps if symbolic assignment is not active or cannot be used (as is the case with devices < V4.2, for example).

On the basis of this configuration, one or more of the following actions should be performed:

- The automatic PROFIdrive telegram setting for a drive object should be activated/deactivated
- The automatic telegram extension for a drive object should be activated/deactivated
- The automatic address adaptation for a drive object should be activated/deactivated
- PROFIdrive telegrams should be configured for drive objects
- The addresses should be set up
- Telegrams should be extended manually.

Note

Telegram synchronization

The telegram synchronization process cannot delete any safety slots for which F-proxy is activated. For this reason, telegram synchronization cannot take place.

- Remove F-proxy, perform telegram synchronization and enable F-proxy again.

Procedure

To do so, proceed as follows:

- In the project navigator, open the entry for the SINAMICS drive unit, e.g. an external S120 or a SIMOTION D.
- In the project navigator, open the **Communication > Telegram configuration entry** under **Control Unit** (SINAMICS standalone CU3xx or **SINAMICS_Integrated)**.
  The **Telegram configuration** dialog is displayed on the tab **IF1 PROFIdrive PZD telegrams**.

The dialog lists all the available drive objects. The possible setting options are described in the following.

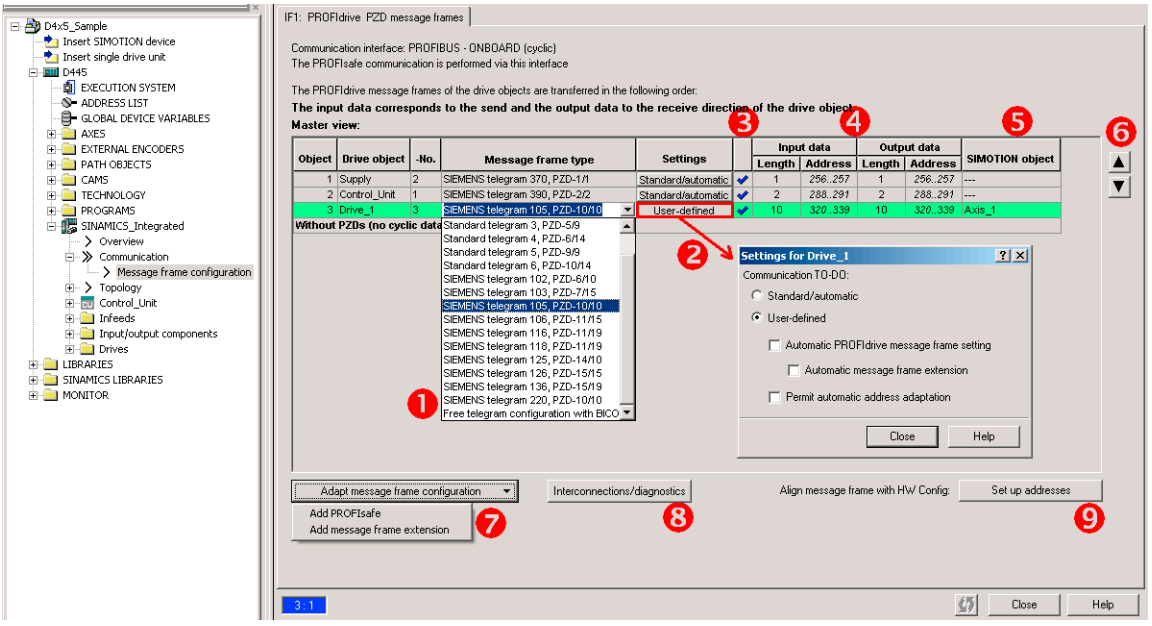


Figure 4-16 Telegram configuration

Table 4-1 Explanation of the figure

Number	Meaning
1	Selection of a telegram • The drive telegrams (telegrams 1 ... 6 and telegram 1xx) are defined in accordance with the PROFIdrive specification and can be selected based on the required functional scope. • You can transfer the signals of the I/Os or the global measuring inputs for the Control Unit, for example, via the telegrams 39x. Telegram 39x is also required for the time-of-day synchronization between SIMOTION and SINAMICS. • Free telegram configuration with BICO allows you to define your own telegram. • Free telegram configuration with p915/p916 (for TM15/17). • Telegrams 37x for control of the infeed.
2	The setting "Standard/automatic" and "User-defined" is visible only if "Use symbolic assignment" is activated. It is generally recommended to use the setting "Standard/automatic". With the "User-defined" setting, the automatic telegram setting, telegram extension and address adjustment can be deactivated and/or activated. • With "Automatic PROFIdrive telegram setting", the telegram is set by the system depending on the configured technology (telegram selection, e.g. for infeed, drive and Control Unit incl. onboard I/O). • With "Automatic telegram extension", the telegram is extended by the system depending on the configured technology (e.g. when the technology data block is activated in the axis configuration). • With "Allow automatic address adjustment", the addresses can be adapted for address displacements, for example, by the system. Address displacements can occur, for example, when a telegram is extended and the adjacent addresses are already occupied by other telegrams. For the TM15 / TM17 High Feature, deactivation of the "Automatic PROFIdrive telegram setting", the "Automatic telegram extension" and the "Automatic address adjustment" is in principle not possible since with these drive objects the telegram is always set up according to the parameterized terminal functionality (DI, DO, output cams, measuring inputs) and cannot be extended. In case the telegrams are to be manually configured for TM15 DI/DO, TM31 and TB30, and BICO is to be interconnected, then "Automatic PROFIdrive telegram setting" and "Automatic telegram extension" must be deactivated. See also Performing message frame configuration in the Online Help.
3	The icons in the status column show the following information:
	 The telegram is configured differently in HW Config. You must align the configuration with HW Config.
	 You are using a predefined standard telegram or free BICO interconnection.
	 You are using an altered standard telegram, which you have extended to include supplementary data.
	 You are using a telegram for which one of the two telegram lengths (I or O) is too long. The drive object cannot process this entry.
4	Length: Displays the size of the telegram component. Address: Address range in HW Config The addresses will be displayed only if they have been defined.
5	Displays the SIMOTION object that is interconnected to the SINAMICS object (e.g. axis or encoder).
6	Change the telegram order. Before the alignment, all drive objects without input/output addresses ("---..---") must be moved behind the objects with valid input/output addresses or those still to be aligned ("???..???").
7	"Manual" adaptation of the telegram configuration (e.g. when additional data, such as a motor temperature, is to be transferred via the telegram)

Number	Meaning
8	Display of the individual control and status words of the associated telegram.
9	Setting up of the addresses (alignment of the addresses with HW Config)

Note

If the telegrams for drive objects change (drives, Terminal Modules, etc.), you must set up the addresses again. The addresses are not updated automatically.

Error correction (symbolic assignment deactivated)

SIMOTION generates further configuration information (FastIO configuration) for the following functions using the 39x telegram:

- Time-of-day synchronization SIMOTION↔ SINAMICS
- Use of onboard I/Os of SIMOTION CU or CX
- Use of output cams and global measuring inputs
- _setDriveObjectSTW system function

If the telegrams are manually defined (symbolic assignment is deactivated), a 39x telegram must be created in the telegram configuration. The telegram must then be compared to the HW Config via "Set up addresses".

If the above functions cannot be used, create a new FastIO configuration. To do this, select the associated SIMOTION CPU in the project tree and right-click to open the "Fast IO" > "Create new configuration" context menu. Compile the project and load it into the CPU. Restart the computer.

The FastIO configuration is also used for the telegram of the Terminals Modules TM15 and TM17 High Feature. Follow the same procedure in the event of problems.

See also

Setting up addresses and message frames - overview (Page 109)

Setting up communication for symbolic assignment (Page 110)

Assigning the TO axis and TO external encoder (Page 89)

4.6 Switching over projects to symbolic assignment

4.6.1 Working with and without symbolic assignment

Description

The symbolic assignment simplifies the configuration of the technological relationships including the communication between controller and drive significantly. The names of the symbolic assignments in plain text is also advantageous for project maintenance.

The symbolic assignment is therefore recommended for new projects as of V4.2 and is automatically activated.

If symbolic assignment is used in a project, telegrams, interconnections and addresses are automatically created by the engineering system per default.

The engineering system sets the "optimum" PROFIdrive telegrams and telegram extensions for the system, makes the required BICO interconnections and determines the addresses.

Activating the symbolic assignment later

It is also possible to change updated projects to symbolic assignment. This must be decided individually, as follow-up work may be required. Particularly in the case of free telegram configurations, comprehensive follow-up work is to be expected.

NOTICE
Possible changes in telegrams and BICO interconnections If symbolic assignment is subsequently activated for a project in which telegrams have already been configured and interconnected, these can be changed together with the BICO interconnections! For this reason, make a backup copy of your project before activating the symbolic assignment.

If symbolic assignment is subsequently activated and telegrams have already been configured and interconnected in the project, the engineering system attempts to identify symbolic assignment for this configuration.

If a symbolic assignment was able to be identified for all communication connections, when the symbolic assignment is activated for the first time, the automatic telegram determination/ extension and address adaptation must then be activated for all drive objects (DOs).

It is not possible to determine a symbolic assignment for all communication connections. This is signaled via a dialog.

The corresponding warnings are then issued in the "Set up assignments" dialog.

Should such a case arise, one of the following measures is required PRIOR TO COMPILATION:

- Option 1: Deactivate automatic for drive object
  In the "Communication" > "Telegram Configuration" drive dialog, the automatic telegram determination/extension and address adaptation for the affected drive objects must be deactivated (user-defined setting, all check marks deselected).
  When the symbolic assignment is activated for the first time, the automatic telegram determination/extension and address adaptation is deactivated for all drive objects (DOs) as standard.
- Option 2: Reconfigure assignments
  The symbolic assignments must be made via the corresponding assignment dialogs of the TO configuration or the address list

Basic information

If necessary, the degree of reconfigurations required is thereby determined, as well as how well "already defined telegram/interconnections" correspond to the settings that the engineering system makes for optimization.

Given the individuality of the "Free telegram configuration with BICO" and the "Telegram extension", it is particularly important to take into account the fact that a symbolic assignment cannot always be determined.

As there is no standard telegram for TB30, TM15 DI/DO and TM31, and "Free telegram configuration with BICO" or "Telegram extension" is thereby applied, these components are particularly affected.

The same applies to the onboard I/Os of a Control Unit (D4x5-2, CX32-2, CU320-2, etc.), if the "Free telegram configuration with BICO" was used here.

Example 1:

If, for example, onboard I/Os of a Control Unit are connected via "Free telegram configuration with BICO", the telegram length, telegram format and interconnection are very individual.

If symbolic assignment is now activated, the image is usually generated on the basis of standard telegram 39x, whereby the telegram length, telegram format and interconnection change.

This then causes disturbances, for example, to the addressing of an onboard I/O that is to be used by SIMOTION.

Example 2:

Telegram 106 is configured for a drive (transfers two encoder values). Only one encoder is used, however.

If symbolic assignment is now activated, optimization of telegram 105 automatically occurs (transfers one encoder value).

NOTICE
Possible changes in telegrams and BICO interconnections If symbolic assignment is subsequently activated for a project in which telegrams have already been configured and interconnected, these can be changed together with the BICO interconnections! For this reason, make a backup copy of your project before activating the symbolic assignment. TB30, TM15 DI/DO and TM31 are especially affected.

See also

Upgrading projects <V4.2 (Page 117)

4.6.2 Using symbolic assignment

Description

Symbolic assignment of SIMOTION and SINAMICS interfaces can be switched on or off as follows. As of V4.2, symbolic assignment is activated by default when new projects are created.

Where projects < V4.2 are being upgraded, symbolic assignment is deactivated by default and needs to be activated if required.

Activating symbolic assignment

1. If the menu entry **Project>Use symbolic assignment** is checked, the function is already activated. If it is not checked, select **Project>Use symbolic assignment**.
   A message dialog opens.
2. Click **OK** to confirm.
   Symbolic assignment is activated and the necessary steps are executed:
   - Assignments between the objects are determined and created symbolically based on the logical addresses
   - Addresses are set up (at the latest automatically before the download)

Note

If you are using symbolic assignment, SCOUT is entirely responsible for managing the telegram between SIMOTION and the drive DO. In other words, SCOUT independently creates telegrams that all contain the necessary signals according to the technological configuration. SCOUT automatically manages the positioning of signals in the telegram and its size; the user is no longer able to influence this process or make any changes.

If you have upgraded a project to V4.2 or started a project initially without symbolic assignment and then select symbolic assignment later on, SCOUT automatically extracts the signal objects contained in the existing telegram configuration and uses them to generate new telegrams automatically based on the pattern you defined. As a specific consequence of this, the number and size of telegram modules (PROFIBUS) or telegram submodules (PROFINET) are bound to change if symbolic assignment is selected (e.g. the number of telegram modules/submodules may be reduced in order to optimize performance).

Deactivating symbolic assignment

The menu item **Project > Use symbolic assignment** is checked if the function is activated.

1. Perform **Project > Use symbolic assignment**.
   A message dialog is displayed indicating that addresses and symbolic names will be set up and checked accordingly.
2. Click **No** to leave symbolic assignment activated.
3. Click **Yes** to deactivate symbolic assignment.

The logical addresses are reassigned priority for the connections between the objects. If this process results in invalid addresses, a message dialog brings this to your attention.

4.6.3 Upgrading projects <V4.2

Description

Should you wish to upgrade projects with a version <V4.2 in order to work with symbolic assignment, you must carry out the following steps where necessary. See also Working with and without symbolic assignment (Page 114).

NOTICE
Possible changes in telegrams and BICO interconnections If symbolic assignment is subsequently activated for a project in which telegrams have already been configured and interconnected, these can be changed together with the BICO interconnections! For this reason, make a backup copy of your project before activating the symbolic assignment. TB30, TM15 DI/DO and TM31 are especially affected.

Procedure

- Open project in SCOUT: The project is opened with deactivated symbolic assignment. In the **Telegram configuration** dialog of the corresponding SINAMICS Control Unit, the dialogs for the symbolic assignment telegram settings are deactivated. As a result, the telegrams, addresses and BICO interconnections that have already been configured are retained.
- Upgrading device version to ≥ V4.2. Only devices as of version V4.2 support symbolic assignment. Devices can be upgraded in HW Config, for example.
- Activating symbolic assignment: When symbolic assignment is activated, the assignments are introduced between axes and drives and are first updated when the project is saved and compiled. The addresses and telegram configuration of SINAMICS DOs continue to remain intact until the automatic telegram configuration is activated in the **Telegram configuration** dialog. In the **Set up assignments** tab, error messages that occur during assignment are output. Using these error messages, you can check and, if necessary, change the TO-DO interconnections.

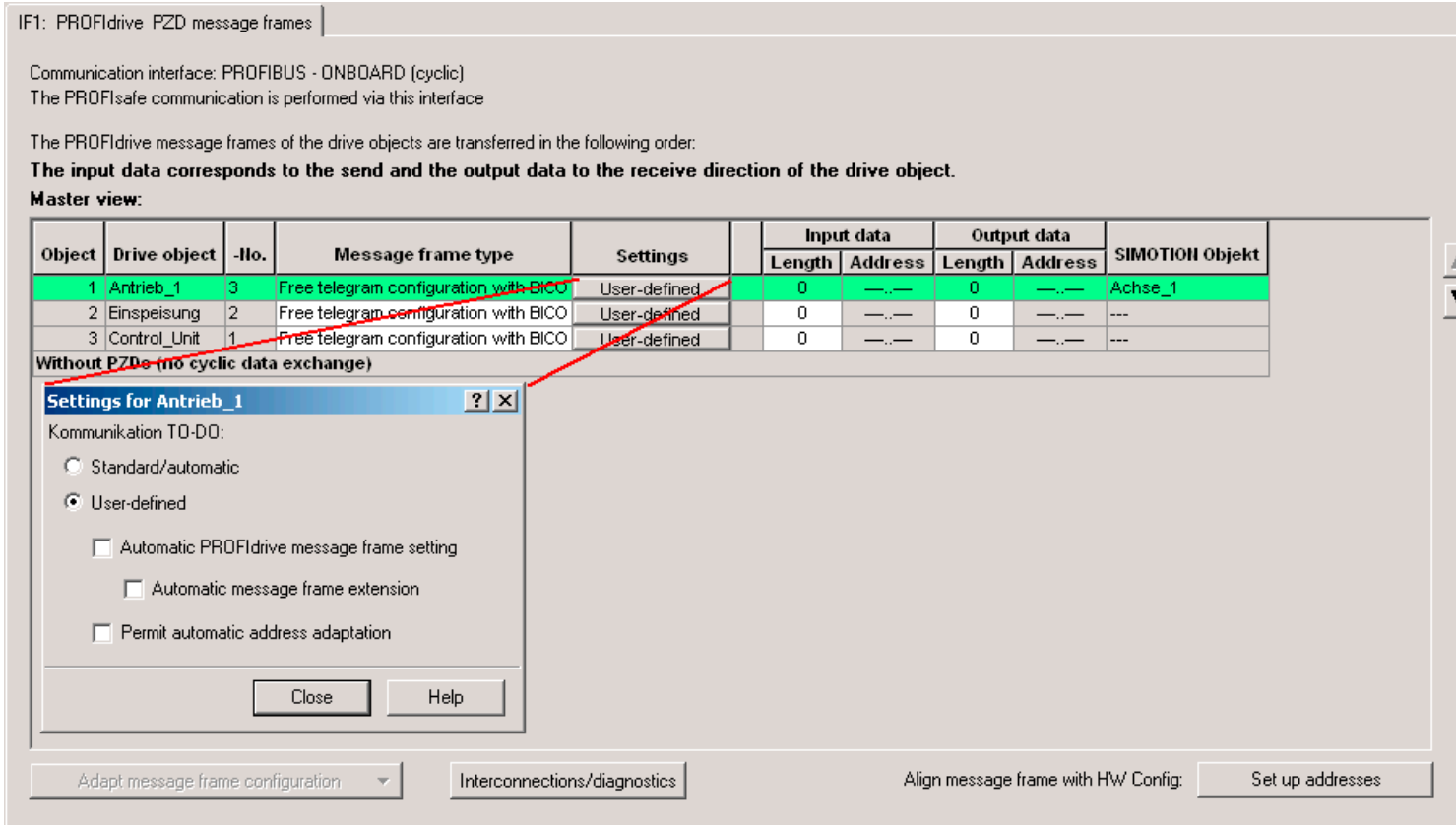


Figure 4-17 Telegram configuration according to high converting

- Assigning SINAMICS DOs: In order to ensure that the SINAMICS DOs are able to participate in the symbolic assignment, the following steps must be carried out in the **Telegram configuration** dialog for each DO:
  - Call dialog **Setting for <DO>** and select the **Standard/Automatic** option when you want to automatically execute the telegram configuration.
    Or
    Call dialog **Setting for <DO>** and select the option **User-defined** if you wish to execute the telegram configuration user-defined, e.g. if you wish to use the setting **Free telegram configuration via BICO** (deselct **Automatic PROFIdrive telegram setting**).
- Setting up communication for symbolic assignment: When you have checked all interconnections, you can run **Project>Setting up communication for symbolic assignment**. All telegrams, addresses and BICO interconnections are then updated via symbolic assignment.

4.6.4 Downgrading to versions <V4.2

Description

If a SIMOTION device is downgraded by replacing a device in HW Config, you must take into account the fact that symbolic assignments are only available as of version V4.2.

If symbolic assignments are used and downgrading to a SIMOTION device <V4.2 occurs, the "Communication for symbolic assignment" must be set up before replacing the device in HW Config. As a result of this, the addresses that are required for a project "without symbolic assignment" are identified.

See also Setting up communication for symbolic assignment (Page 110) for more information.

4.6.5 Safety data block for a SINAMICS device version less than V4.x

Description

Problems occur with older SIMOTION modules and SINAMICS CUs that have a firmware version less than V4.x in the following combination:

- Used STANDARD telegram 4/6 or SIEMENS telegram 103/106 (each with and without a technology data block)
- Safety data block
- Servo drive
- Double-encoder system

Occurring problems

Upgraded projects:

The following message is output during subsequent activation of the symbolic interconnection for upgraded projects:

"Address/value of I/O variable 'Axis_x: TypeOfAxis.TechnologicalData.DriveSafetyExtendedFunctionsInfoDataIn.logAdress' (address: PIB xxx[x]) is invalid or is not supported by the hardware. (If required, perform FastIO "Create new configuration".)"

Newly created projects:

With newly created projects (and therefore activated symbolic assignment), the SIDB cannot be created by the system because the limited velocity must be interconnected to a double word.

Message output at activation of SINAMICS Safety Integrated Extended Functions on the TO axis:
"SINAMICS_Integrated: Drive_x: Not enough space to set up the telegram extension".

Procedure

Upgraded projects:

In the "Telegram configuration", do not set the settings to "Standard/automatic" and deactivate the automatic PROFIdrive telegram setting and the automatic telegram extension, or make the appropriate settings.

If the system has deleted the telegram extension due to the "Standard/automatic" setting, make the settings as described above and interconnect the safety data block again (parameter 9733[0] to the first free word after the safety data block and leave the following word empty).

Newly created projects:

Deselect "Standard/automatic" in the telegram configuration settings and deactivate the automatic PROFIdrive telegram setting and the automatic telegram extension. Interconnect parameter 9733[0] to the first free word after the safety data block and leave the following word empty.

With upgraded and new projects:

Go to the address list and set the view to "All addresses". At the appropriate axis ("Axis name".Actor.SIDB), enter the address PIxxx at which the safety data block is located. For example, SIEMENS telegram 106 has 15 PZD and therefore the following applies:

Logical start address of the drive +30 / when using the technology data block +32

For further information, see also AUTOHOTSPOT in the TO Axis Function Manual.

Programming with Technology Objects 5

5.1 Definitions

The following section provides an introduction to the motion components primarily from the perspective of the ST programming language. This mainly involves system functions, system variables and configuration data. Some definitions are provided below:

- System functions are functions used for the system management. They provide access to technology-neutral functionality of the device. System functions are always available.
- A technology object (TO) represents a technological functionality (for example, positioning an axis, assigning parameters for an output cam) in the SIMOTION user program.
- Technology object functions or technology commands are language commands provided by the individual technology objects, i.e. functions for a technology object.
- A technology package contains one or more technology object types, from which the respective TO instance is generated with <Insert technology object>.
- System variables and configuration data are attributes of the technology objects and the basic system. You can use system variables to assign parameters for technology objects and the basic system or to read their status.

Note

You will find additional information about the basics, configuration and programming of motion control technology and, in particular, technology objects, in the SIMOTION Motion Control <Technology Objects> Function Manuals.

The specified topics are only briefly described in this documentation.

5.2 Programming technology objects (TOs)

5.2.1 Using technology functions in a program

In order to use technology functions (TO functions), you must select a technology package once. Technology objects (TOs) and, thus, TO functions are contained in the technology package. Formally, these are functions with a function name, input parameters and, usually, a return value. Below is an overview of the codes and components of the technology objects and TO functions except for the input parameters, which are described in the **Function parameters of the technology functions** section.

The following is presented from the ST programming perspective. For LAD/FBD and MCC programming see the respective manuals.

Selecting a technology package

You use the following command to select the technology package:

`USEPACKAGE` *tp-name*

Here *tp-name* is the name of the technology package (see next table).

Table 5-1 Technology packages

Technology package	Description of contents
CAM	• Basic motion control commands, positioning, gearing and discontinuous synchronous operation (cam) • Commands for external encoders, measuring inputs, output cams, and cam tracks
PATH[1]	As for TP CAM plus commands for path interpolation
CAM_EXT	As for TP PATH plus commands for supplementary technology objects (fixed gear, addition object, formula object, sensor, controller object)
TControl	Commands for temperature controllers

[1] Available as of version V4.1.

Technology object (TO) codes

A technology package provides technology objects (TOs); they are instantiated in the SIMOTION SCOUT. An instantiated TO is addressed (referenced) in the program by means of its name.

Codes of technology functions (TO functions)

TO functions are primarily commands that execute specific actions on the technology object, which is why they are also referred to as TO commands. You can also use TO functions in user-defined FBs. For more information about the format rules of user-defined FCs and FBs, please refer to the SIMOTION ST Programming Manual. The following table shows an overview of the formal codes of the TO functions.

Table 5-2 Codes of the TO functions

Code	Description
Name	All TO function names (*_enableAxis* in the example) are defined identifiers in the SIMOTION system that always begin with _ (underscore). To maintain a distinction between these TO functions and user-defined FBs and FCs, you should not create source file sections beginning with the _ character.
Input parameter	When called, TO functions can contain one or more input parameters and always supply a return value to the call location. Output parameters are not supported. For additional information, see Input parameters (Page 482).
Return value	All commands normally return a double-precision integer (DINT data type). This indicates whether the command was successfully transferred to the system and processed normally (return value of zero) or whether an error occurred (return value other than zero).

Example

If the application requires you to enable a virtual axis, you could program a source file like the one shown in the following figure.

The following conditions must be fulfilled:

- An instance of a position axis or speed axis has been created in SIMOTION SCOUT as a virtual axis called Axis_1.
- The *myPos* program has been assigned to *MotionTask_1*, for example. The **Activation after StartupTask** option has been selected in the task configuration of *MotionTask_1*.
- The source file has been downloaded to the target system.

After the CPU has changed to RUN mode, virtual axis *Axis_1* will issue an enable. You can verify the status of the axis enable in system variable *Axis_1.control*.

Table 5-3 Example for using TO functions in a program

```
INTERFACE
    USEPACKAGE CAM;
    PROGRAM myPos;
END_INTERFACE
IMPLEMENTATION
(* The following program must be assigned to a MotionTask. The "Activation
after StartupTask" option must be selected in the task configuration. *)
    PROGRAM myPos
        VAR
            retVal : DINT;
        END_VAR
        // Axis is enabled for positioning.
        retVal := _enableAxis (
            axis := Achse_1,
            // TO instance identifier
            nextCommand := WHEN_COMMAND_DONE,
            // Condition for program advance.
            commandId := _getCommandId() );
            // Unique command ID
    END_PROGRAM
END_IMPLEMENTATION
```

Note

Use the long form for passing parameters (with value assignment) described in **Function parameters of the technology functions**. This form is clearer and more flexible. For example, the program code therefore remains compatible with new function parameters for a function extension.

You will find tips on efficient use of parameters in system functions in **Error sources and efficient programming**.

See also

Function parameters of the technology functions (Page 126)

5.2.2 Sample program with namespace option

You can use the optional *namespace* add-on AS to define a name space. You can then also access types, variables, functions and function blocks in the technology package that have the same name as those in the ST source file.

The following example shows how to select the CAM technology package, assign it the namespace Cam1, and use the namespace:

Table 5-4 Example of selecting a technology package and using a namespace

```
INTERFACE
    USEPACKAGE CAM AS Cam1;
    USES ST_2;
    FUNCTION function1;
END_INTERFACE

IMPLEMENTATION
    FUNCTION function1 : VOID
    VAR_INPUT
        p_Axis : posAxis;
    END_VAR
    VAR
        retVal : DINT;
    END_VAR

    retVal:= Cam1._enableAxis (
            axis := p_Axis,
            nextCommand := Cam1.WHEN_COMMAND_DONE,
            commandId := _getCommandId() );
    END_FUNCTION
END_IMPLEMENTATION
```

Note

If a namespace is defined for an imported library or technology package, this must **always** be specified if a function, function block or data type from this library or technology package is being used. See above example: Cam1._enableAxis, Cam1.WHEN_COMMAND_DONE.

See also

Function parameters of the technology functions (Page 126)

Efficient programming - overview (Page 598)

5.2.3 Differences between cyclical and sequential programming

Cyclic tasks

Cyclic tasks (such as the BackgroundTask) will be started by the system after their completion or automatically after a defined time frame. The values of the static variables of the assigned programs are retained. Cyclic tasks have a time monitoring and a defined error response should the time monitoring be exceeded. This means the code contained in cyclic tasks must perform its tasks quickly and efficiently. Tasks with a waiting character (for example, wait for the positioning of an axis) can only be performed in several call cycles of the cyclic task. This means TO system commands must normally be called with the IMMEDIATELY step enabling condition for the nextCommand parameter. The subsequent call cycles must then check the result of the system command for successful processing or error. This procedure is also called asynchronous execution.

Sequential tasks

After start, sequential tasks (e.g. MotionTasks) are executed once and then terminated. At each start, all local variables of the assigned programs are initialized; see Influence of the compiler on variable initialization (Page 322) for variable initialization. Because sequential tasks do not have any time monitoring, they can attain a run time of any length. Sequential tasks are subject only to the control of the application. This means the application can start, stop, pause and resume tasks. The code contained in sequential tasks processes tasks successively, where the following task is normally performed only when the previous task has been completed. For example, an axis is first released and then homed. This means TO system command calls should be performed using the WHEN_COMMAND_DONE step enabling condition for the nextCommand parameter. The system function returns to the calling sequential task only when the command has been processed. This is also called synchronous execution.

General Procedure

In general, it is better to program sequential sequences in MotionTasks. The differences between a sequential programming in a MotionTask and the cyclical programming in the BackgroundTask are:

- As part of the sequential processing of a the MotionTask, one and only one (but nevertheless linked) step enabling condition can be awaited at any point in time. The step enabling condition is checked with high priority in the interpolator cycle clock. To reduce the response time for continuing the MotionTask, its priority can be increased temporarily (-> WAITFORCONDITION).
- A normally cyclical program checks in each cycle a number of signal states and step enabling conditions. This heavily loads and extends the cycle time. The advantage compared with the sequential type of programming lies in the parallel processing of requests and sequences.

Synchronous and asynchronous command execution

Synchronous command execution:

Further processing of the program is suspended at the command until the command has been fully executed (preferably in synchronous tasks).

Asynchronous execution:

The function is activated via a command and program processing continues (preferably in cyclic tasks) while the function is still being executed by the system.

For more information, please read Transition and step enabling condition (Page 131).

5.2.4 Function parameters of the technology functions

The function parameters of the TO functions are:

- *mandatory*, i.e. they must be specified, for example, the TO instance and the target position for positioning;
- *optional predefined*, i.e. they can be specified; otherwise, a default setting for the system is activated, for example, IMMEDIATELY for the step enabling condition;
- *optionalUserDefault*, i.e. they can be specified, but if they are not, a user-defined default behavior takes effect (programmable or configurable in the associated userDefault variable). For information about the use of parameter values in motion control programs, refer to the documentation for the technology objects (TOs).

The TO instance name must always be specified for the TO commands as they cannot be preset by the system. In our **Example of variables to be specified**, you must specify which axis is to be activated (axis := myaxis).

Short and long form of the function parameter specification

The function parameters can be specified with or without function parameter identifiers.

The following variants are supported (see IEC1131/3 2nd Edition, 11/98):

- **Short form**
  "Call by value" as call with fixed arrangement and number of input variables corresponding to the function declaration
- **Long form**
  "Call by name" as call with variable arrangement and number of input variables

Short form

The function parameters (parameter identifiers) are omitted and only the current data values (parameter values) are specified. An assignment operator is not permitted.

All parameter values (even optional ones) must be separated by commas *in the correct order*!

Only a few system functions (see **Functions for the runtime measurement of tasks**, **Task control commands** and **Functions for the message configuration**) require the short form of parameter transfer when called. This is specified explicitly for the relevant functions.

Long form

The transfer parameters are assigned to the formal operands. Transfer parameters with assigned default values can be specified optionally, but this is not essential.

Note

Use the described long form (with value assignment). This form is clearer and more flexible. For example, the program code therefore remains compatible with new function parameters for a function extension.

Mandatory rules for specifying function parameters

The following table shows the rules you must follow when specifying function parameters.

Table 5-5 Rules for specifying function parameters

Rule	Example
Short form	
All function parameters must be transferred.	In the **Example of variables to be specified** the actual values *myAxis* (variable) and *IMMEDIATELY* (value) are assigned to the function parameters *axis* and *nextCommand*.
The order of the function parameters according to the function declaration must be maintained.	-
Long form	
Only function parameters that are not optional must be specified. The default value is used for parameters that are not specified.	In the **Example of variables to be specified**, the following is also possible: *_enableAxis (axis:=myAxis)*
The order of the function parameters is optional.	In the **Example of variables to be specified**, the following is also possible: *_enableAxis (nextCommand:= IMMEDIATELY, axis:=myAxis)*
Several function parameters	
Several function parameters are separated by commas.	In the **Example of variables to be specified**: *_enableAxis (axis:=myAxis, nextCommand := IMMEDIATELY)*
Actual values as variables	
The advantage is that you can reuse the source file sections that use these variables. For example, a user-defined function block (FB) is to call a TO function with variable axis identifiers as function parameters. It would be awkward to have to call the function several times using constant axis names and would also mean that you could not reuse the FB. You can also use other actual values as variables.	In the **Example of variables to be specified**, the user-defined function block *posFB* defines the TO instance *myAxis* (*myAxis:PosAxis*). When you call a TO function from this FB, the TO instance is used as actual values (*Axis := myAxis*). The values for the TO instance are derived not from the user-defined FB, but from the program called by this FB with *myFB (myAxis := Axis)*. For the reasons named, you can also call the FB and, thus, the TO function with *myAxis := Axis2* or *myAxis := Axis3*, etc. This means you can use the FB in all programs of the ST source file and in programs of other source files with the aid of the export function.
Actual values as values (enumerators)	

Rule	Example
If you enter actual values as values (*next Command: = IMMEDIATELY* in the example), you must usually choose a value from various specific states (only for enumerators). Enumerators are a derived data type. You will find basic information about enumerators in Chapter "User-defined data types" in the ST Programming Manual. You can also specify direct values for other data types.	In the **Example of variables to be specified**, enumerators IMMEDIATELY and WHEN_COMMAND_DONE are available for function parameter *nextCommand* in TO function *_enableAxis*. The compiler rejects other values during compilation of the program.
Parameters for the transition and step enabling condition	
With many motion commands, you can specify when the TO function is executed on the technology object and when the next statement is processed (see **Transition and step enabling condition**)	-
Parameters for command identification	
All motion commands must contain a parameter for the command identification. This allows you to track the status of the command execution (see **Diagnosis of the command execution**).	-

Note

All system data types for enumerations (in system functions and system variables) begin with the word *Enum*. For example, function parameter *nextCommand* has enumeration data type *EnumNextCommandEnable* (with values *IMMEDIATELY* or *WHEN_COMMAND_DONE*).

All system data types for structures (in system functions and system variables) begin with the word *Struct*.

If you do not begin user-defined data types with character string *Enum* or *Struct*, names cannot overlap. For a detailed description of of the name spaces, see ST Programming Manual.

Appendix D of this manual contains all reserved identifiers of the ST (Structured Text) programming language, the system functions, the system blocks and the SIMOTION devices.

The reserved identifiers of the SIMOTION technology packages can be found in the List Manuals for the SIMOTION technology packages.

Reference and value specification for motion variables

The parameters of motion variables (e.g. velocity, acceleration) are defined using a reference parameter and a value parameter.

- The reference parameter specifies which system variable the motion variable to be transferred references and, if required, how the following value parameter is to be interpreted.
  The identifier of a reference parameter is formed from the identifier of the motion variable with the suffix *Type*, e.g. *velocityType*.
- The value parameter specifies:
  - For reference parameter = DIRECT: The numerical value of the motion variable.
  - For reference parameter = USER_DEFAULT: The scaling factor of the default value stored in the system variable.

  The value parameter has no significance for other reference parameter settings.
  The identifier of a value parameter is the identifier of the motion variable, e.g. *velocity*.

Table 5-6 Frequent reference parameters and effect of the associated value parameters

Reference parameter	Value parameter	Value of the motion variable
DIRECT	Absolute value	Content of the value parameter (with the unit of the motion variable specified during the configuration of the technology object). See example in the table below.
USER_DEFAULT	Relative value [%]	Default setting * value parameter / 100 The default settings for the motion variable are stored in a system variable. See example in the table below.
ACTUAL	–[1]	Current actual value
CURRENT	–[1]	Current setpoint of the interpolator
EFFECTIVE	–[1]	Last programmed value
[1] Value parameter is ignored		

Table 5-7 Example of velocityType (reference parameter) and velocity (value parameter)

velocityType	velocity	Value of the motion variable
DIRECT	(No details)	100 [configured unit]
	50.0	50 [configured unit]
	200.0	200 [configured unit]
USER-DEFAULT	(No details)	100% of the system variable value[1]
	50.0	50% of the system variable value[1]
	200.0	200% of the system variable value[1]
[1] For motion commands of an axis: system variable *userDefault.velocity*		

Data type of TO instance variables

You can use TO instance variables, for example, for object specification in TO function calls.

Note

New TO data types (as of V4.0) begin with a "_" to prevent confusion with user variables.

You must first declare the variables, however, and you must choose from a predefined selection of data types (see table for **Data types for technology objects**). In our **Use of TO functions in the program** examples in this section (**Example of variables to be specified**), the data type is *PosAxis*, because you want to create TO instance variables for position axes. The *Cam* (*discontinuous synchronous operation*) technology package you have selected contains the *PosAxis* data type for the *Position Axis* technology object.

Table 5-8 Data types of technology objects (names of technology objects)

Technology object	Data type	Contained in the technology package
Drive axis	DriveAxis	CAM, PATH[1], CAM_EXT
External encoder	ExternalEncoderType	CAM, PATH[1], CAM_EXT
Measuring input	MeasuringInputType	CAM, PATH[1], CAM_EXT
Output cam	OutputCamType	CAM, PATH[1], CAM_EXT
Cam track	_CamTrackType	CAM, PATH[1], CAM_EXT
Position axis	PosAxis	CAM, PATH[1], CAM_EXT
Following axis	FollowingAxis	CAM, PATH[1], CAM_EXT
Following object	FollowingObjectType	CAM, PATH[1], CAM_EXT
Cam	CamType	CAM, PATH[1], CAM_EXT
Path axis[1]	_PathAxis	PATH[1], CAM_EXT
Path object[1]	_Pathobjecttype	PATH[1], CAM_EXT
Fixed gear	_FixedGearType	CAM_EXT
Addition object	_AdditionObjectType	CAM_EXT
Formula object	_FormulaObjectType_	CAM_EXT
Sensor	_SensorType	CAM_EXT
Controller object	_ControllerObjectType_	CAM_EXT
Temperature channel	TemperatureControllerType	TControl
General data type, to which every TO can be assigned	ANYOBJECT	

As of Version V4.1

Variables of the technology object (TO) data type are initialized by default with the value TO#NIL. You can use this to check whether a valid TO was assigned to a variable; see **Example of variables to be specified**.

Example of variables to be specified

An example of variables to be specified with a technology object data type (an example of optional use was described in **Data types for technology objects**) follows. A reusable FB will be written, which will enable any axis. Since the axis name is variable, you must define a variable with the data type of the axis.

The axis to be enabled is provided when the FB is called. This is verified for safety reasons. If no TO is available (TO#NIL), execution of the FB is interrupted.

Table 5-9 Example of variables to be specified with a technology object data type

```
// ...
FUNCTION_BLOCK posFB
    VAR_INPUT
        myAxis: posAxis;
    END_VAR

    VAR_OUTPUT
        // Return value of the TO function,
        // also output parameter of the FB
        return_value: DINT := -1;
    END_VAR
    // Check for valid TO
    IF myAxis = TO#NIL THEN RETURN; END_IF;
        // Example of call with variable of data type of TO
        return_value := _enableAxis (
        axis:= myAxis, // TO function
        nextCommand:= IMMEDIATELY, //optional
        commandId    := _getCommandId() );
END_FUNCTION_BLOCK
PROGRAM Example
    VAR
        myFB: posFB;
    END_VAR
    myFB (myaxis := Axis1);
    // Name is created on start-up // in the SIMOTION SCOUT.
    myFB (myaxis := Axis2);
    // Name is created on start-up // in the SIMOTION SCOUT.

END PROGRAM
//...
```

5.2.5 Transition and step enabling conditions

Fundamentals for the processing of a TO function (command execution)

The TO functions are transferred as commands to the technology function for execution. The TO executes or activates these commands in the processing cycle clock, which was specified during their configuration (e.g. IPO cycle clock).

The outputCam, measuringInput, externalEncoder and cam technology objects have a direct command execution. A new command on the same technology object supersedes a command which is active there.

In addition to commands for direct command execution, motion commands can be issued on the following technology objects: driveAxis, posAxis, followingAxis, and followingObject. The technology objects have structure elements for command management.

An example of the command management for axes is described below.

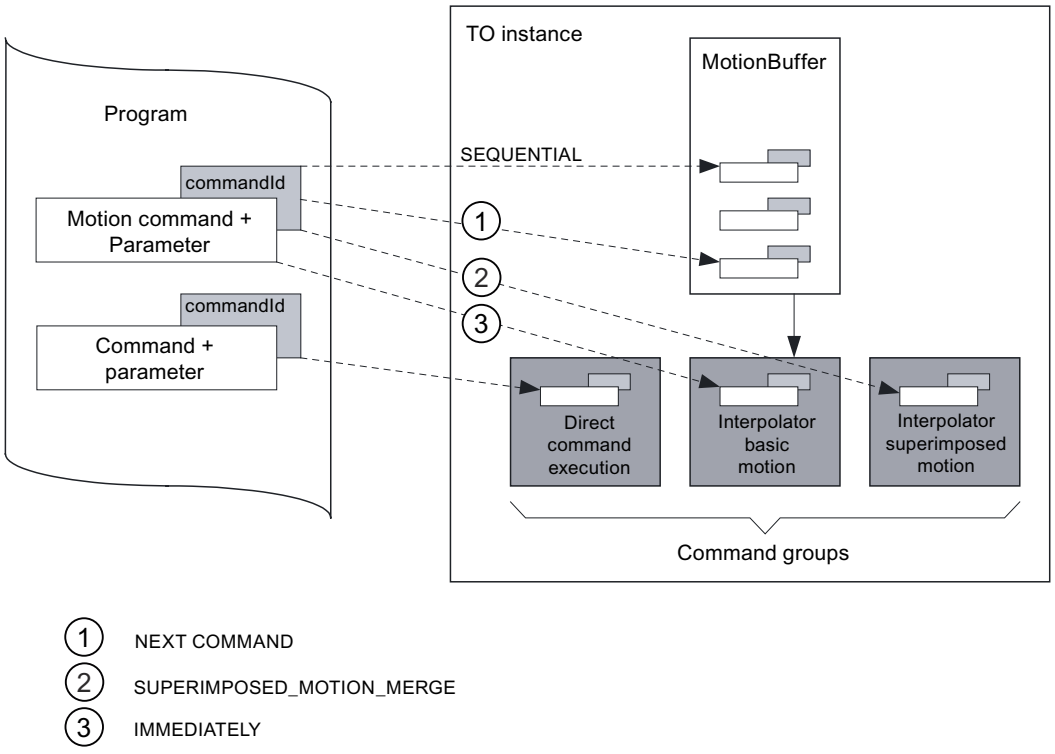


Figure 5-1 Command execution on the axes

The MotionBuffer stores motion commands that are executed sequentially by the interpolator. The number of motion commands that can be stored in the motion buffer is defined via the configuration data *TypeOfAxis.DecodingConfig.numberOfMaxbufferedCommandId*. Multiple motion commands can therefore be issued on the technology object, irrespective of the execution status of the active command.

A **commandId** is assigned to each command when it is issued. This is stored in the command and provides a reference to the issued command.

The commands are assigned to command groups. The pending commands in the existing command groups are processed in parallel by the interpolator. Certain commands become

active on the technology object immediately and form a queue in the motion buffer; other commands, i.e. superimposing commands, take effect immediately.

Note

The behavior of the command groups and command buffers, e.g. MotionBuffer, is TO-specific. Thus, for example, you cannot call the _enableaxistorquelimitpositive and _enableaxistorquelimitnegative commands simultaneously within one IPO2 cycle clock. Only one of the commands will be executed.

An exact description of command groups and command buffers can be found in the TO manuals, e.g. in the "TO Axis Electric / Hydraulic, External Encoder" function manual.

Transition behavior of the currently active motion command

You specify the transition behavior of the currently active motion command on the technology object in the TO function with the parameter *MergeMode*. Here you specify, how the TO function is placed in the command execution sequence on the technology object, or to which command group it is assigned.

Table 5-10 Frequent merge modes

Merge mode	Description
IMMEDIATELY	The motion specified in the command becomes active immediately; already active motions are replaced, pending commands/motions are aborted.
NEXT_MOTION	Execute after the active motion and delete further pending commands/motions.
SEQUENTIAL	Attach to previous commands/motions.
SUPERIMPOSED_ MOTION_MERGE	Superimposed motion on the axis is possible in addition to the basic motion.

Command step enabling condition

A step enabling condition can be programmed for TO functions that are executed in the interpolator. It specifies when the next statement in the program source is executed.
The *nextCommand* parameter is used for this purpose. Refer to the SIMOTION Cam Technology Package, System Functions List Manual (reference list) for the permissible conditions for each command.

The main difference is between asynchronous and synchronous command execution:

- Asynchronous command execution:
  The TO function is transferred to the technology object and the program continued immediately. The *nextCommand* = IMMEDIATELY is set for this.
  In this case, the application must ensure that TO functions are issued only once and checkback signals are evaluated explicitly by scanning the axis or command status (see **Diagnosis of the command execution**).
  Example: see the following two figures
  This method of motion control is referred to as *cyclic programming*. It is permitted in all system tasks and is intended especially for the programming of cyclic tasks.
  The asynchronous command execution is the default setting when parameter *nextCommand* is not specified.
- Synchronous command execution:
  The TO function together with the parameterization for the step enable are transferred to the technology object and the called task stopped. The technology object executes the function and calls for program execution to be resumed as soon as the specified step enabling condition is satisfied or the command has been aborted. For this, *nextCommand* is set to the desired step enabling condition.
  Example: see **Asynchronous program execution (sequential programming)**.
  This method of motion control is referred to as *sequential programming*. It is supported especially by the MotionTasks.
  Programming command sequences in cyclic tasks leads to task timeouts and therefore to runtime errors.

Table 5-11 Example of asynchronous program execution (cyclic programming) - Part 1

```
INTERFACE
USEPACKAGE CAM;
    PROGRAM ProgramCycle;
END_INTERFACE

IMPLEMENTATION

PROGRAM ProgramCycle
    VAR
        boStartCommand : BOOL; // Command - issue command
        boCommandStarted : BOOL; //Auxiliary variable - command issued
        boCommandDone : BOOL; // Auxiliary variable - command executed
        i32Ret : DINT; // Return value of system functions
        sCommandId : CommandIdType; // CommandId
        // Return value - _getStateOfAxisCommand
        sRetCommandState : StructRetCommandState;
        // Instance of the system FB for the edge detection
        r_trig_1 : R_TRIG;
    END_VAR

    r_trig_1 (boStartCommand); // Call the edge detection

    IF r_trig_1.q THEN
        // Request for a system-wide unique commandId
        sCommandId := _getCommandId ();
        // Register commandId at the TO
        // --> Diagnostics of end or abort of command possible
        i32Ret := _bufferAxisCommandId (
        axis := Axis_1,
        commandId := sCommandId );
        // Evaluation of return value of system function
        // ...
        // Issuing of a command - motion with USER_DEFAULT values
        i32Ret := _move(
        axis := Axis_1,
        nextCommand := IMMEDIATELY,
        commandId := sCommandId );
        // Evaluation of return value of system function
        // ...
        // Auxiliary variables for coordination of command execution
        boCommandStarted := TRUE;
        boCommandDone := FALSE;
//-----------------------------------------------------------------------
// Continuation follows
```

Table 5-12 Example of asynchronous program execution (cyclic programming) - Part 2

```
// Continuation
//-----------------------------------------------------------------------
    ELSIF boCommandStarted AND NOT boCommandDone THEN
        // Query command execution status
        sRetCommandState := _getStateOfAxisCommand(
        axis := Axis_1,
        commandId := sCommandId );

    IF sRetCommandState.functionResult = 0 THEN
        IF sRetCommandState.commandIdState = EXECUTED THEN
            // Command has been executed (completed)
            boCommandStarted := FALSE;
            boCommandDone := TRUE;
            // Remove registered commandId on TO
            i32Ret := _removeBufferedAxisCommandId(
            axis := Axis_1,
            commandId := sCommandId );
        END_IF;
    ELSE
        // Error handling for _getStateOfAxisCommand function call
        // ...
        ;
    END_IF;

    ELSIF boCommandDone THEN
        // Execution after command execution
        // ...
        ;
    END_IF;

 // Further user program as of here
 // ...

END_PROGRAM

END_IMPLEMENTATION
```

Table 5-13 Example of synchronous program execution (sequential programming)

```
INTERFACE
    USEPACKAGE CAM;
    VAR_GLOBAL
        g_boCommandStarted : BOOL; //Auxiliary variable - command issued
        g_boCommandDone : BOOL; // Auxiliary variable - command executed
        END_VAR
        PROGRAM ProgramSequential;
END_INTERFACE

IMPLEMENTATION
    PROGRAM ProgramSequential
        VAR
            i32Ret : DINT: // Return value of system function
        END_VAR;
        g_boCommandStarted := TRUE;
        g_boCommandDone := FALSE;
        // Statements executed before the motion.
        // ...
        i32Ret := _move(
        axis := Axis_1,
        nextCommand:= WHEN_MOTION_DONE,
        commandId:= _getCommandId () );

        // Statements executed after the motion.
        // ...
        // Evaluation of return value of system function
        // ...

        g_boCommandStarted := FALSE;
        g_boCommandDone := TRUE;

    END_PROGRAM
```

See also

CommandID overview (Page 447)

5.2.6 Command execution diagnostics

Command identification – commandId

When a system function issues a command, a commandId is transferred. While the command is being executed by the technology object, it stores the commandId in the command, thus identifying the command.

A project-wide unique commandId is obtained with the *_getCommandId* system function. This ensures that no further command with the same commandId exists in the system (unique reference to the command).

Table 5-14 Example of the use of the commandId

```
//...
VAR
    myCommandId : CommandIdType;
END_VAR
//...
// Save unique ID
myCommandId := _getCommandId ();
// Execute function with ID
myFC := _pos (axis := myAxis,
        position        := position_1,
        nextCommand        :=IMMEDIATELY,
        commandId        :=myCommandId);
//...
```

The description how you can track the execution status of a command with the aid of the commandId follows.

System functions for scanning the command/execution status

Technology objects, on which several commands are issued for execution, have *_getStateOf...Command* system functions (e.g. *_get StateOfAxisCommand*). The return value of data type *StructRetCommandState* provides information on the execution status of a motion command by the interpolator in the *EnumCommandIdState* component.

With *EnumCommandIdState* = ABORTED, the abort reason is specified in the component *abortId* (data type DINT). The values correspond to the reason in the technological alarm "30002 Command aborted".

Status check after completion or abort of a command

Once a command has been aborted or has finished executing, it is normally deleted from the internal command management system of the technology object. As a result, it is not possible to diagnose the command status *End* or *Abort* by calling the system functions indicated above.

To enable the status of a command to be scanned even after it has been aborted or its execution is complete, the commandId of the relevant command must be made known to the internal command management of the technology object. This is performed using the *_buffer...CommandId* system function (e.g. _bufferAxisCommandId).

After the *End* or *Abort* status has been evaluated, the commandId must be explicitly removed from the command management of the technology object. This is performed via the system function *_removeBuffered...CommandId* (e.g. *_removeBufferedAxisCommandId*).

Example 1	The _buffer... command is not issued, and the _pos command for which the query is to be made is already finished.
Result	Because a matching command for the CommandId specified in the _getStateOf...CommandId was not found (_pos is already finished), NOT_EXISTENT ('commandId' is not known or command is already finished) is returned.
Example 2	The _buffer... command is issued, and the _pos command for which the query is to be made is already finished.
Result	The _pos command is no longer found, but the result was stored in the CommandId buffer. The result is now either EXECUTED (command processing finished) or ABORTED (command processing aborted).

The size of the CommandID buffer is limited and can be set, for example, for the axis with configuration data item *TypeOfAxis.DecodingConfig.NumberOfMaxBufferedCommandId*. Thus, you can notify the TO regarding the maximum number of commands it has to manage simultaneously.

On the STOP-RUN transition, the buffered Command IDs will be deleted. The CommandID buffer is then empty.

Example: see **Asynchronous program execution (cyclic programming), Part 1 and Part 2**.

The behavior of the "buffer and removeBuffer" commands is the same in all TOs that support this functionality (exceptions: names of commands and name of the configuration data item for the buffer size).

Note

The description applies analogously to the external encoder, as well.

Status check after resetting a technology object

The CommandID can be buffered in such a way that it is not deleted when a technology object is reset. It is then also available after resetting a technology object.

- To do this, call the TO function *__buffer...CommandId* with the parameter *deleteCommandIdWithReset = NO*. The commandId can then only be deleted explicitly with the command *_removeBuffered...CommandId*.
- If you call the *_buffer...CommandId* TO function with the *deleteCommandIdWithReset= YES* (default setting) parameter, the commandId is also deleted with the *_reset...* function (e.g. *_resetAxis*) when resetting the technology object.

See also

CommandID overview (Page 447)

5.2.7 Identifiers of technology object instances

The identifiers of technology object instances are defined during their configuration in SIMOTION SCOUT. They must be defined uniquely within a SIMOTION device.

In general, you can call the instance of a technology object with its identifier.

However, if the same identifier is defined as data type, variable, function or function block in ST source files or if a global device variable or I/O variable of the same name was created, these identifiers cover the technology object instance.

You can use the predefined name space *_to* so that you can still access the technology object instance, for example to access its system variables or configuration data (see **Name spaces** in the ST Programming Manual). Place the name space identifier in front of the corresponding name, separated by a period, for example *_to.to-name*.

If you want to access the instance of a technology object on another SIMOTION device, place the name of the SIMOTION device in front of the instance identifier, separated by a period, for example *dev-name.to-name* or *dev-name._to.to-name*.

If the identifier of a device is covered, you can use the predefined name space *_project*, for example *_project.dev-name.to-name* or *_project.dev-name._to.to-name*.

Note

With a project-wide unique identifier for the technology object instance, you can use the predefined name space *_project* also for identifying the instance.

5.2.8 Determining the TO name at runtime (as of V4.4)

The system function *_getObjectName* as string is used to determine the configured TO name at runtime.

E.g., you can use it to determine the name of an object causing an error and to display it on an HMI.

When the function is called in the TechnologicalFaultTask, you can transfer TSI#toInst directly from the TaskStartInfo in the function's input parameter.

5.2.9 Conversion of TO data types

Type conversions within the hierarchical data types

The *driveAxis*, *posAxis*, *followingAxis* TO data types are hierarchically structured by the functional scope.

- A position axis (*posAxis* data type) contains the functionality of a speed-controlled axis (*driveAxis* data type).
- A following axis (*followingAxis* data type) contains the functionality of a position axis (*posAxis* data type) and, therefore, also of a speed-controlled axis (*driveAxis* data type).

Type conversions are only possible within these hierarchical data types and with the general ANYOBJECT technology object data type.

Note

Other type conversions are not possible (for example, between a measuring input and following object).

Implicit type conversion

Variables (with TO data type) or TO instances can be assigned to the following variables without specifying a conversion function:

- Variables of a lower hierarchy TO data type:
  - *followingAxis* to *posAxis* or *driveAxis*
  - *posAxis* to *driveAxis*
- Variables of general ANYOBJECT TO data type

See the table below.

Table 5-15 Example of implicit type conversion

```
// The following TO instance (axis) is configured in the project navigator:
fol_axis_real as a following axis

VAR
    drv_axis1 : driveAxis;
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR

drv_axis1 := pos_axis1;
any_obj1  := fol_axis_real;...
```

Explicit type conversion

The *anyObject_to_Object* type conversion function is used to assign variables (with a TO data type) to variables with higher hierarchy TO data types.

A requirement for this is that the source variable (with the hierarchically lower TO data type) must refer to a TO instance that at least corresponds hierarchically to the TO data type of the target variable (see example in the following table).

Table 5-16 Example of successful type conversions

```
// The following TO instances (axes) are configured in the
 // project navigator:
 // pos_axis_real as position axis
 // fol_axis_real as a following axis

VAR
    drv_axis1 : driveAxis;
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR

// Implicit type conversions
drv_axis1 := pos_axis_real;
any_obj1  := fol_axis_real;

// Successful type conversions

pos_axis1 := anyObject_to_Object (in := drv_axis1);
 // Type conversion successful,
 // because drv_axis1 refers to a position axis.

pos_axis1 := anyObject_to_Object (in := any_obj1);
 // Type conversion successful,
 // because any_obj1 refers to a following axis.

//...
```

In the event of a failed type conversion, the value TO#NIL is assigned to the target variable:

Table 5-17 Example of a failed type conversion

```
// The following TO instance (axis) is configured in the
 // project navigator:
 // drv_axis_real as a drive axis

VAR
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR

// Implicit type conversions
any_obj1  := drv_axis_real;

// Type conversion failed
pos_axis1 := anyObject_to_Object (in := any_obj1);
        // Type conversion is not required,
        // because any_obj1 refers to a speed-controlled axis.
        // pos_axis1 has the value TO#NIL.
```

5.2.10 System variables

You can use system variables to assign parameters for technology objects and the basic system or to read their status.

Syntax of system variables

System variables are queried using a structure access with the following syntax. The following table shows the significance of the individual syntax components.

[_to.]TO name.variable.component or

[_device.]variable.component

Table 5-18 Syntax components of system variables

Syntax component	Meaning
TO name	*TO name* stands for the name of a technology object (TO), e.g. of an axis. You have performed one of the following: • Inserted the technology object in SIMOTION SCOUT. • Or declared a variable of the data type of this technology object in the program. You will find a list of all the data types for technology objects in **Function parameters of the technology functions**.
_to	The optional word *_to* identifies the predefined name space for technology objects. It should be used to unambiguously specify a technology object with *TO name*. Also see **Name spaces** in the ST programming manual.

Syntax component	Meaning
_device	The optional word *_device* identifies the predefined name space for device-specific variables (system variables of the SIMOTION devices, I/O variables, global device variables). It should be used to unambiguously specify the variable as system variable of the SIMOTION device. These system variables are also available when no technology object is loaded. Also see **Name spaces** in the ST programming manual.
Variable	*Variable* stands for the name of the system variable as found in the list of all system variables (see List Manual for the SIMOTION technology package system variables).
Component	*Component* stands for the part of the structure you want to query. This can be an additional structure that contains further components, as well. The depth of the structure depends on the system variable and can be zero.

Using system variables

Read access to a system variable is always possible. A system variable can be assigned to a variable in two ways:

- With a value assignment (see also **Value assignments** in the ST programming manual). This means ExecutionFaultTask is called in the event of an error (please also refer to *Reading out the substitute value or last values* at the end of the chapter). For more information on the error reaction, see Accesses to system variables (Page 156).
- With the *_getSafeValue* and *_setSafeValue* system functions (see _getSafeValue function (Page 420), _setSafeValue function (Page 422), and Accesses to system variables and inputs/outputs (Page 419)). In this case it is possible to program the desired response in the event of an error.

The use in an expression or as a parameter in a function or function block is also possible. In this case, ExecutionFaultTask is also called in the event of an error (see Accesses to system variables (Page 156)).

If a system variable can be written is specified in the List Manual (reference list) of the system variables of the technology objects (TO) in the following entry:

- Effective: *immediately* (can be read and written to)
- Effective: *read only* (can only be read).

If a system variable can be written to, it can be assigned a value in two ways:

- With a value assignment (see also **Value assignments** in the ST programming manual). This means ExecutionFaultTask is called in the event of an error (please also refer to *Reading out the substitute value or last values* at the end of the chapter). For more information on the error reaction, refer to Access errors to system variables and configuration data, as well as I/O variables for direct access (Page 156).
- With the system function _setSafeValue (see function _setSafeValue (Page 422)). In this case it is possible to program the desired response in the event of an error.

TO#NIL is a special variable value. You can use this value to check whether a valid technology object is present (see the example in **Function parameters of the technology functions**).

Note

For performance reasons, you should only access system variables when absolutely necessary. Instead, save their contents in a local variable of the same data type. Local variable accesses need much fewer resources because less processor time is used. For more information, see **Efficient programming**.

Also note that a source of error can be comparing REAL variables, LREAL variables, and system variables (for example, axis position) with each other, see **Compare REAL or LREAL variables**.

Note

As of SIMOTION Kernel V4.4, system variables can be accessed from synchronous user tasks.

Note

System variables saved ONLINE cannot be saved with "Save to memory card (Ram2Rom)" or "Save in the engineering project (load configuration data to PG)" is not possible.

So that values of system variables can also be saved in the engineering project and on the memory card, the default value of system variables must be changed OFFLINE and then loaded to the target system per download and saved. See Storage concept in the target system (Page 547).

Scope of system variables

1. System variables, for example, the status flag, may only exist for just one IPO cycle clock.
2. All system variables have the documented **Update** property.
3. If you want to query the status of a task with a lower cycle time, the status should be linked with the following OR status. The OR operation ensures that all states of the application are used. The operation ensures that the subsequent ENUMS of the status flag are grouped.

Examples of system variables

You want to check the axis position and dynamic axis state of *Axis1*.

Requirements:

- You have created Axis1 in SIMOTION SCOUT or have defined and initialized it in the program, e.g. using the *myAxis* variable of *PosAxis* data type.
- You have defined variables in the program for recording the axis position and the dynamic axis state. The data type of these variables must match the data type of the variables to be checked, e.g.:

```
VAR
act_pos : LREAL;
```

```
act_motionState : EnumAxisMotionState;
END_VAR
```

Example of accessing a system variable using a structure element of an elementary data type:

```
act_pos := Axis1.positioningState.actualPosition;
```

PositioningState is the system variable and *actualPosition* is the structure element of data type LREAL that will be queried.

Example of querying a system variable with an enumeration element:

```
act_motionState := Axis1.motionStateData.motionState;
```

motionStateData is the system variable and *motionState* is the structure element of enumerator data type *EnumAxisMotionState* that will be queried.

Initialization of system variables

System variables are not reinitialized as a rule when you download the project; their actual values are not reset to the initial values. In general, system variables are only reinitialized when the SIMOTION device is restarted.

Substitute value or the last value of system variables at TO restart or where TO is deactivated (as of V4.1)

The access to system variables is also possible at RESTART of the TO or at activated TO, without the system going to STOP. Instead of reading out the system variables using the _getSaveValue function, you can configure the following by means of an entry in the config data (restart.behaviorInvalidSysvarAccess) to enable direct access:

- Read out last value (LAST_VALUE = default)
- Read out default value (=value when loading the project; DEFAULT_VALUE)
- Go to STOP (STOP_DEVICE)

Exception

Variables that deliver the current TO status, also return the correct status at RESTART. This affects the system variable restartActivation, which you can access via _getSafeValue.

Note

TO system variables can be written to during a TO restart or where a TO is deactivated (apart from STOP_DEVICE). The values will be applied or are effective after the RESTART. If writing system variables exceeds the limits, the CPU will go into STOP mode. As of V4.2, writing can take place outside the valid limits. The limit value will be accepted in this case.

See also

Function parameters of the technology functions (Page 126)

5.2.11 Configuration data

Configuration data defines the basic functionality of a technology object. It is normally set during the configuration of the technology object with SIMOTION SCOUT and for the most part cannot be modified during runtime.

A major part of this configuration data can, however, be modified while the program is running. Whether or not a configuration data item can be modified during runtime is specific to each configuration data item and is documented in the List Manual for SIMOTION technology package configuration data.

Depending on the configuration data item, the following options are available:

- Cannot be modified online:
  This configuration data can only be modified during configuration of the technology object with SIMOTION SCOUT.
- Can be modified online, effective after restart:
  This configuration data can be changed by variable access from the user program. The change does not take effect until the technology object is restarted (see **Resetting a technology object**).
- Can be modified online, immediately effective:
  This configuration data can be changed by variable access from the user program. Change is immediately effective.

Note

If you make a change to configuration data which only takes effect after a TO restart, subsequent changes to "effective immediately" configuration data will also only take effect after the TO restart.

Note

As of SIMOTION Kernel V4.4, configuration data can be accessed from synchronous user tasks.

Note

Configuration data changed ONLINE can be saved on a memory card with "Copy current data to RAM" and subsequent "Copy from RAM to ROM" or saved with "Load to PG" in the engineering project. See Storage concept in the target system (Page 547).

Reading the configuration data

You can read each configuration data item of a technology object and assign it to a variable, for example:

- The value saved in the SIMOTION device's RAM (can be read in the "Next value" column in the expert list)
  To do so, use the syntax: *TO-name*.setconfigdata.*config-date*.
- Value currently in effect on the technology object
  To do so, use the syntax: *TO-name*.activeconfigdata.*config-date*.

See example in the table below.

Only access to individual variables of the configuration data is permitted.

Table 5-19 Read access to a configuration data item of a technology object

```
VAR
    lreal_var : LREAL;
    drive_var : driveaxis;
END_VAR

lreal_var :=
    drive_var.setconfigdata.TypeOfAxis.MaxJerk.maximum;
    // Read access to saved value
lreal_var :=
    drive_var.activeconfigdata.TypeOfAxis.MaxJerk.maximum;
    // Read access to value currently in effect
```

A configuration data item can be assigned to a variable in two ways:

- With a value assignment, such as the example in the previous table (see also **Value assignments** in the ST Programming Manual). This means *ExecutionFaultTask* is called in the event of an error (please also refer to *Substitute value or last value* at the end of the chapter). For more information about the error reaction, see Errors when accessing configuration data (Page 420).
- With the *_getSafeValue* system function (see function _getSafeValue (Page 420)). In this case it is possible to program the desired response in the event of an error.

The use in an expression or as a parameter in a function or function block is also possible. In this case, the *ExecutionFaultTask* is called in the event of an error.

Modifying configuration data at runtime (online modification)

Configuration data is modified online by a simple variable write access to the value stored in the RAM. To do so, use the syntax:
TO-name.setconfigdata.*config-date*.

The effectiveness (adoption as the current value in effect on the technology object) is determined by the configuration data item (effective immediately / effective after restart).

See example of **write access to a configuration data item**.

Only access to individual variables of the configuration data is permitted.

Table 5-20 Write access to a configuration data item of a technology object

```
VAR
    lreal_var : LREAL;
    drive_var : driveaxis;
END_VAR

drive_var.setconfigdata.TypeOfAxis.MaxJerk.maximum :=
    200000.0;
    // Write access to saved value
```

A configuration data item can be assigned a value in two ways:

- With a value assignment, such as the example in the previous table (see also **Value assignments** in the ST programming manual). This means *ExecutionFaultTask* is called in the event of an error (please also refer to *Substitute value or last value* at the end of the chapter). For more information about the error reaction, see Errors when accessing configuration data (Page 156).
- With the system function *_setSafeValue* (see function _setSafeValue (Page 422)). In this case it is possible to program the desired response in the event of an error.

In addition, there is the option to control the activation via a system variable.

Use technology object system variable *activationModeChangedConfigData* to define when the modified configuration data is to take effect:

- If *activationModeChangedConfigData = ACTIVATE_CHANGED_CONFIG_DATA* is set, the modified data is set immediately active. If the system variable is set to this value, data collected up to this point is also activated.
- If *activationModeChangedConfigData = COLLECT_CHANGED_CONFIG_DATA* is set, the modified data is collected.
  They are activated as a body as soon as this system variable is set to *ACTIVATE_CHANGED_CONFIG_DATA*.
  The collected, modified configuration data can be deleted (without activation) by calling the corresponding technology object system function (e.g. *_resetAxisConfigDataBuffer*).

Note

When *activationModeChangedConfigData*:= *ACTIVATE_CHANGED_CONFIG_DATA*, it takes a certain amount of time for a modified configuration data item to take effect after it has been written.

In particular, when several configuration data items are changed simultaneously, timeouts can occur in the tasks.

Note

If you want to change several configuration data items at the same time, it is advisable to collect them first with *activationModeChangedConfigData = COLLECT_CHANGED_CONFIG_DATA* and then activate them as a body in a sequential task using the *ACTIVATE_CHANGED_CONFIG_DATA* setting.

Configuration data changed at runtime can be saved on card and in the ES project, see Memory access (Page 550).

Note

When accessing configuration data from the user program (e.g. in a general FB), make sure that the respective configuration data is also available depending on TO type.

Access to configuration data for TO references and axis arrays

Access to configuration data for TO references is possible. To access configuration data of an axis array you must use an intermediate variable. If you attempt to use axis arrays directly you will get a compiler message that an intermediate variable should be used.

Example

```
VAR_TEMP
    myPosAxis  :posaxis;
END_VAR
myPosAxis      :=Pos[2]

myPosAxis.setconfigdata.TypeOfAxis.MaxJerk.maximum :=
    200000.0;
```

Because you are accessing the axis directly via a reference, no further assignment is necessary to access the value. You do not have to reassign pos[2] to myPosAxis.

Note

Write operations to configuration data are possible during a TO restart or where a TO is deactivated (apart from STOP_DEVICE). The values will be applied or are effective after the RESTART.

Substitute value or the last value of configuration data at TO restart or where TO is deactivated (as of V4.1)

Configuration data can also be accessed when a RESTART of the TO is performed or where the TO is deactivated without the system going to STOP. Instead of reading out the configuration data using the _getSaveValue function, you can configure the following by means of an entry in the configuration data (restart.behaviorInvalidSysvarAccess):

- Read out last value (LAST_VALUE = default)
- Read out default value (=value when loading the project; DEFAULT_VALUE)
- Go to STOP (STOP_DEVICE)

If writing configuration data exceeds the applicable limits, the CPU will go to STOP. As of V4.2, writing can take place outside the valid limits. The limit value will be accepted in this case.

5.2.12 Resetting a technology object

⚠ CAUTION
Resetting a technology object aborts the current motion without an error message.

To integrate configuration data that require a restart of the technology object, you must reset the technology object. The procedure depends on the *restart.restartActivationSetting* configuration data item of the technology object:

- For the setting *restart.restartActivationSetting* = RESTART_BY_COMMAND:
  The technology object can only be restarted by means of the corresponding system function (e.g. *_restartAxis*). To do so, set the parameter *activateRestart* = ACTIVATE_RESTART.
- For the setting *restart.restartActivationSetting*= RESTART_BY_SYSVAR_AND_COMMAND:
  The restart can be performed in two ways:
  - By calling the corresponding system function (e.g. *_restartAxis*). By setting parameter *activateRestart* = ACTIVATE_RESTART.
  - By assigning a value to the technology object system variable: *restartActivation*= ACTIVATE_RESTART. The system variables are initialized, the technology object loses all of its status information, such as axis homed.

  The restart is always performed asynchronously. After a successful restart of the technology object, this system variable has the value NO_RESTART_ACTIVATION.

5.2.13 Use of technology packages in libraries

Libraries can also contain TO functions and accesses to the system variables of a technology object.

You specify the SIMOTION device and the technology object for which the library is being compiled in the object properties for the library as follows:

1. Select the library in the project navigator.
2. Select the **Edit > Object properties** menu command.
3. Then select the **Technology packages** tab.
4. Select the SIMOTION devices (including the version number) and the technology packages for which the library is to be compiled.

Note

To compile a project without errors, observe the rules governing the selection of SIMOTION devices and technology packages in the following table!

You can also specify the technology package in the library's ST source files if you want (with the USEPACKAGE command), however, this is not necessary.

Table 5-21 Selecting devices and technology packages in a library

Selection	Description
Device-independent	You must also select: • The technology packages • The version number of the selected technology packages **Note:** 1. The library is compiled without reference to a SIMOTION device or a version of the SIMOTION kernel. **For this reason, the following must not be used:** – System functions of SIMOTION devices – System variables of SIMOTION devices – Version-dependent system functions – Configuration data of the technology objects 2. The library is compiled precisely to the version selected. The use of system functions or variables which are not available in the selected version will result in a compilation error. 3. If a device-independent library is to be made available for another version it must be copied and inserted under a different name. This copy must be recompiled with a different version reference.
SIMOTION device including version (multiple selection possible)	Only those technology packages are displayed that are available for all of the selected devices. **Note:** 1. The library is compiled for all of the selected devices and technology packages (of the selected device versions). 2. The use of system functions or variables which are not available for one of the selected devices, or the technology package of the respective device version, will result in a compilation error. 3. The library can only be used for the selected devices and technology packages. When you use the library in an ST source file, the following is therefore checked: – Whether the library is compiled for the SIMOTION device (including version) that contains the importing ST source file. – Whether the technology package set on the SIMOTION device and specified in the ST source file with the USEPACKAGE command corresponds to the one in the library. Any inconsistencies will result in compilation errors.

5.3 Response to faults and events

5.3.1 Evaluating faults and events

The SIMOTION system has various execution levels for scheduling programs. One of these execution levels is the interrupt execution level, which is started in response to specific events, such as errors.

Options for responding to faults and events

There are two classes of events that can start tasks in the interrupt execution level:

- When events are user-defined, they are called UserInterruptTasks; the programs assigned to these tasks are also user-defined.
- If the events are system or technology-related (system errors or technology objects), we refer to SystemInterruptTasks. The following table lists the available SystemInterruptTasks. Messages generated by the system are known as alarms. The reaction is defined in the alarm configuration (see below).

Table 5-22 Specified SystemInterruptTasks

SystemInterruptTask	Called for the following events
ExecutionFaultTask	Execution errors in programs
PeripheralFaultTask	Process and diagnostic alarms from I/O modules
TechnologicalFaultTask	Alarms, warnings, and notes from technology objects
TimeFaultBackgroundTask	Timeout for the BackgroundTask
TimeFaultTask	Timeouts for TimerInterruptTasks

Note

Message generation is another system feedback option. User-assigned messages can be used in programs, e.g. if certain events occur (tank empty, etc.). For more information,
see Programming messages (Page 471).

Alarm configuration

You can define the system behavior for technological alarms in SIMOTION SCOUT (alarm configuration). You can choose between:

- STOP: Transition to STOP mode (all system and user tasks are stopped.)
- STOP U: Transition to STOP U mode (only user program tasks are stopped.)
- START TechnologicalFaultTask: Starts the associated SystemInterruptTask
- NONE: No response

Each alarm has a default response. Details for the alarm configuration can be found at Error handling for technology objects (Page 179).

When you select **START TechnologicalFaultTask**, you must assign a program to the TechnologicalFaultTask that responds to the associated alarm.

Besides a configurable response to program execution, alarms also have a reaction in the technology object (see SIMOTION Motion Control Technology Objects function manuals).

5.3.2 Execution errors in programs

You set the response to execution errors in programs in the task configuration (see **Configuring the execution system**).

This affects the following operations, for example:

- Invalid operation with floating-point numbers (Page 155) (e.g. logarithm of a negative number)
- Incorrect type conversions
- Division by zero
- Violation of array limits

The possible error responses are listed in the following table.

Table 5-23 Error response in case of execution errors in the program

Error response	Task type	Description
CPU to STOP	All tasks	SIMOTION device switches to STOP mode and the ShutdownTask is started.
ExecutionFaultTask	Sequential (non-cyclic)	The ExecutionFaultTask is started. The task in which the error occurred, is aborted; the aborted task can be dispatched again by the user.
	Cyclic	The ExecutionFaultTask is started. The task in which the error occurred is aborted. When the ExecutionFaultTask is finished, the SIMOTION device switches to STOP mode; the ShutdownTask is started.

See also

Specifications for the configuring (Page 320)

5.3.3 Error during operations with floating-point numbers (FPU exceptions)

Invalid floating-point numbers

The data types for REAL and LREAL floating-point numbers are implemented with their bit patterns according to IEEE 754. Accordingly, the following bit patterns for invalid floating-point numbers are also supported:

- Signaling NaN (NaNs): Invalid bit pattern (Not a Number), which triggers an error (FPU exception) on each operation.
- Quiet NaN (NaNq): Invalid bit pattern (Not a Number), which triggers an error (FPU exception) only on certain operations.
- Infinity: Bit pattern for + infinity or – infinity.

Note

You can create an NaN or infinity from the corresponding bit pattern, e.g. with system function DWORD_TO_REAL, BigByteArray_to_AnyType or LittleByteArray_to_AnyType.

Note

When a floating-point number is displayed in the engineering system (e.g. in the symbol browser of SIMOTION SCOUT), no distinction is made between a signaling and a quiet NaN.

FPU exceptions

The operations with floating-point numbers are also implemented according to IEEE 754. If any of the indicated errors for the operations below occur, an FPU exception will be triggered:

- Any operation with a signaling NAN (NaNs).
- Addition: Both addends are infinity but have different signs.
- Subtraction: Minuend and subtrahend are infinity and have the same sign.
- Multiplication: One factor is 0 and the other is infinity.
- Division:
  - Both operands are 0 or infinity.
  - Divisor is 0.
- Modulo division (x MOD y): x = infinity or y = 0.

- The argument of a system function is outside the domain, e.g.:
  - SQRT: The radicand is < 0.
  - LN, LOG: The argument is ≤ 0.
  - EXPT or Operator "**": Base is ≤ 0:
    Exceptions as of Version 4.1 of the SIMOTION Kernel:
    Base is < 0 and exponent has an integer value.
    Base is = 0 and exponent is > 0.
  - SIN, COS, TAN: The argument is infinity.
  - ASIN, ACOS: Argument is > 1
- Conversion of a floating-point number to an integer or its assigned bit data type with the corresponding system function (e.g. LREAL_TO_DINT, REAL_VALUE_TO_DWORD):
  - Range of target data type is exceeded.
  - Argument is a signaling or quiet NaN (NaNs or NaNq).
  - Argument is infinity.
- Comparison operations:
  - At least one operand is a signaling or quiet NaN (NaNs or NaNq).
  - At least one operand is infinity.
- Range is exceeded for operations with valid floating-point numbers

The behavior specified for the processing errors in programs (Page 154) will be carried out. If the ExecutionFaultTask is called, then TSI#executionFaultType = _SC_INVALID_FLOATING_POINT_OPERATION, see TaskStartInfo of the ExecutionFaultTask (Page 165).

Note

No error (no FPU exception) is triggered:

- For operations with quiet NaN (NaNq), if not mentioned explicitly above.
  For example, the addition of a valid floating-point number to a quiet NaN (NaNs) produces the same quiet NaN (NaNs).
- For operations with + infinity or - infinity, if not mentioned explicitly above.
  For example, the addition of a valid floating-point number to + infinity produces +infinity again.

5.3.4 Access errors to system variables and configuration data, as well as I/O variables for direct access

This section describes the behavior when errors occur while accessing system variables, configuration data or I/O variables with the usual methods (using the variable identifier in an expression or variable assignment), see also TaskStartInfo of the ExecutionFaultTask (Page 165).

Errors in system variables and configuration data:

As of V4.1 SP2/V4.1 SP3, system variables and configuration data can also be accessed when a RESTART of the TO is performed or where the TO is deactivated without the system entering STOP mode.

You can configure the following by means of an entry in the configuration data (restart.behaviorInvalidSysvarAccess):

- Read out last value (LAST_VALUE = default)
- Read out default value (=value when loading the project; DEFAULT_VALUE)
- Go to STOP (STOP_DEVICE)
  The ExecutionFaultTask is started, and the additional error response depends on the task type (sequential or cyclic) in which the error occurs (see the following table).
  Response at start of ExecutionFaultTask for incorrect access to system variables and configuration data

Task type	Description
Sequential (non-cyclic)	The ExecutionFaultTask is started. The task in which the error occurred is aborted; the aborted task can be dispatched again by the user.
Cyclic	The ExecutionFaultTask is started. The task in which the error occurred is aborted. When the ExecutionFaultTask is finished, the SIMOTION device switches to STOP mode; the ShutdownTask is started.

Writing of system variable values outside the applicable limits is not affected by the configuration data referred to above (e.g. STOP_DEVICE). As of V4.2, the limit value is written. See also System variables (Page 143), Configuration data (Page 147), General information on accessing system variables and inputs/outputs (Page 419) or Errors when accessing system data with _get/_setSafeValue (Page 193).

Definition of DEFAULT_VALUE and LAST_VALUE

- DEFAULT_VALUE
  The DEFAULT_VALUE is the configured value. This is the value transferred during a download (system variables and configuration data).
- LAST_VALUE
  - Configuration data
    The last value can be the configured value. The value will then be equivalent to the value shown under "Current value" in the expert list.
    OR
    The last value can be the last configured value. The value will then be equivalent to the value shown under "Next value" in the expert list.
    Either of these values can be output, depending on the configuration data entered.
  - System variables
    With system variables, the last value is the value which is currently set and effective.

Errors for I/O variables (direct access to inputs and outputs)

The error response is specified when the I/O variables are defined (see **Direct access and process image of the cyclical tasks** in the ST programming manual).

- CPU stop: The ExecutionFaultTask is started. The SIMOTION device then switches to STOP mode; the ShutdownTask is started.
- Substitute value: The substitute value specified when the I/O variable was defined is adopted, and the task is continued.
- Last value:
  With read access (to inputs or outputs): The last valid value is applied, and the task is continued.
  With write access (to outputs): The value is written to the variable. However, it will not be active at the output until the output becomes available again. The task is continued.

In specific cases it is necessary to respond differently to errors, to deviate from the defined error response, or to avoid errors by performing queries beforehand. The functions _getSafeValue (Page 420) and _setSafeValue (Page 422), and getInOutByte (Page 426) serve this purpose. The functions *_getSafeValue* and *_setSafeValue* are very time intensive.

5.3.5 Errors when generating the process image

This section describes the behavior when I/O access errors occur during updating of the process image with the assigned task. Possible causes:

- The I/O module is not present.
- The I/O module is switched off.
- The connection to the I/O module is missing or faulty.
- The I/O module is signaling an error.

The behavior is as follows:

- For the process image of the cyclic tasks (see **Direct access and process image of the cyclic tasks** in the programming manuals)
  The error response is specified when the I/O variables are defined:
  - CPU stop: For response information, see the following table.
  - Substitute value: The substitute value specified when the I/O variable was defined is adopted, and the cyclic task is continued.
  - Last value:
    Process input image (reading of inputs): The value of the process image at the address is not changed; the cyclic task is continued.
    Process output image (writing to outputs): The value only takes effect at the output with the address when the output is available again; the cyclic task is continued.
- For the fixed process image of the BackgroundTask (see **Accesses to the fixed process image of the BackgroundTask** in the programming manuals):
  The error response depends on whether direct access was defined at the same address using I/O variables:
  - **No** direct access is defined: Error response is always CPU STOP; for response information, see the following table.
  - Direct access is defined: The error response specified in the definition of the I/O variables does apply (see above, like process image of cyclic tasks).

Table 5-24 Error response during process image update for CPU STOP response

Event	Description
Error occurs	1. An incoming message is generated once. 2. If no program is linked to the PeripheralFaultTask, the SIMOTION device goes to STOP mode, and the ShutdownTask is started. 3. Otherwise: – The PeripheralFaultTask is started once **immediately** (rather than in the next IPO cycle clock): *TSI#interruptId* = _SC_IMAGE_UPDATE_FAILED. *TSI#logBaseAdrIn* or *TSI#logBaseAdrOut* contains the address at which the error occurred. See TaskStartInfo of the PeripheralFaultTask (Page 167). – Process input image: The substitute value is assigned to the value of the process image at the address. Process output image: Value will not take effect at the output with the address until the output becomes available again. – The cyclic task in which the error occurred is continued.
Error persists	• No additional messages are generated. • The PeripheralFaultTask is not restarted. • Process input image: The value of the process image at the address is not changed. Process output image: Value will not take effect at the output with the address until the output becomes available again.
Error disappears	1. An outgoing message is generated once. 2. The PeripheralFaultTask is started once **immediately** (rather than in the next IPO cycle clock): *TSI#interruptId* = _SC_IMAGE_UPDATE_OK, see TaskStartInfo of the PeripheralFaultTask (Page 167). 3. The cyclic task is continued.

5.3.6 Using Taskstartinfo

Important information about starting the task is stored in the Taskstartinfo for each task, e.g.:

- Start time of task
- For the TechnologicalFaultTask: the triggering instance of the Technology Object and the alarm number,
- For the TimeFaultTask: TimerInterruptTask that caused the timeout error.

Within a task, you can query the relevant Taskstartinfo of this task. To do this, use the **TSI#<info>** system variable; where **<info>** is the particular information to be queried. The content and scope of the TaskStartInfo and the associated system variables depend on the relevant task. This is described in the following sections.

Details on the TaskStartInfo of the

StartupTask (Page 161)
MotionTasks (Page 162)
BackgroundTask (Page 162)
TimerInterruptTasks (Page 162)
UserInterruptTasks (Page 163)
SynchronousTasks (Page 163)

SystemInterruptTasks
- ExecutionFaultTask (Page 165)
- PeripheralFaultTask (Page 167)
- TechnologicalFaultTask (Page 175)
- TimeFaultBackgroundTask (Page 176)
- TimeFaultTask (Page 177)

ShutdownTask (Page 177)

The TaskStartInfo query is used mainly for SystemInterruptTasks, see example for using the TaskStartInfo of the TechnologicalFaultTask (Page 178).

See also

Evaluating in the user program (Page 191)

5.3.6.1 TaskStartInfo of the StartupTask

Table 5-25 TaskStartInfo of the StartupTask

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use

TSI#taskType	:	ENUM...	Type of task
			As of version V4.5 of the SIMOTION Kernel
			Data type: EnumTaskType
			Value: STARTUP
			`TYPE EnumTaskType`
			`  STARTUP  // (1)`
			`  MOTION  // (2)`
			`  BACKGROUND  // (3)`
			`  SERVO_SYNCHRONOUS  // (4)`
			`  IPO_SYNCHRONOUS  // (5)`
			`  PWM_SYNCHRONOUS  // (6)`
			`  INPUT_SYNCHRONOUS  // (7)`
			`  POST_CONTROL  // (8)`
			`  SYSTEM_INTERRUPT  // (9)`
			`  TIMER_INTERRUPT  // (10)`
			`  USER_INTERRUPT  // (11)`
			`  SHUTDOWN  // (12)`
			`END_TYPE`

5.3.6.2 TaskStartInfo of the MotionTasks

Table 5-26 TaskStartInfo of the MotionTasks

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: MOTION

5.3.6.3 TaskStartInfo of the BackgroundTask

Table 5-27 TaskStartInfo of the BackgroundTask

Designation	:	Data type	
TSI#startTime	:	DT	Time of cycle control point
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is non-equidistant and cyclic)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is non-equidistant and cyclic) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: BACKGROUND

5.3.6.4 TaskStartInfo of the TimerInterruptTasks

Table 5-28 TaskStartInfo of the TimerInterruptTasks

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Time of cycle control point
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms

Designation	:	Data type	Meaning
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: TIMER_INTERRUPT

5.3.6.5 TaskStartInfo of the UserInterruptTasks

Table 5-29 TaskStartInfo of the UserInterruptTasks

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: USER_INTERRUPT

5.3.6.6 TaskStartInfo of the SynchronousTasks

Table 5-30 TaskStartInfo of the SynchronousTasks

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of task (= cycle clock with which the task runs synchronously)
TSI#currentTaskId	:	StructTaskId	TaskId of task

Designation	:	Data type	Meaning
TSI#cycleTime	:	TIME	Configured cycle time of task in ms • ServoSynchronousTask: Position control cycle clock • ServoSynchronousTask_fast: Fast position control cycle clock • IPOsynchronousTask: Interpolator cycle clock (IPO) • IPOsynchronousTask_fast: Fast interpolator cycle clock IPO • IPOsynchronousTask_2: Interpolator cycle clock (IPO_2) • PWMsynchronousTask: Pulse-width modulation cycle clock • InputSynchronousTask_1: Input1 cycle clock • InputSynchronousTask_2: Input2 cycle clock • PostControlTask_1: Control1 cycle clock • PostControlTask_2: Control2 cycle clock
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs • ServoSynchronousTask: Position control cycle clock • ServoSynchronousTask_fast: Fast position control cycle clock • IPOsynchronousTask: Interpolator cycle clock (IPO) • IPOsynchronousTask_fast: Fast interpolator cycle clock IPO • IPOsynchronousTask_2: Interpolator cycle clock (IPO_2) • PWMsynchronousTask: Pulse-width modulation cycle clock • InputSynchronousTask_1: Input1 cycle clock • InputSynchronousTask_2: Input2 cycle clock • PostControlTask_1: Control1 cycle clock • PostControlTask_2: Control2 cycle clock SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Values: • SERVO_SYNCHRONOUS • IPO_SYNCHRONOUS • PWM_SYNCHRONOUS • INPUT_SYNCHRONOUS • POST_CONTROL

5.3.6.7 TaskStartInfo of the ExecutionFaultTask

Table 5-31 TaskStartInfo of the ExecutionFaultTask

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use

Designation	:	Data type	Meaning
TSI#executionFaultType	:	UDINT	Type of execution error in user program • _SC_DIVISION_BY_ZERO (= 500) Error during division or modulo division of integers (data type ANY_INT or ANY_BYTE): Divisor is 0. • _SC_INVALID_FLOATING_POINT_OPERATION (= 501) Error during operations with floating-point numbers (FPU exceptions) (Page 155) • _SC_ARRAY_BOUND_ERROR_READ (= 502) Array boundaries exceeded during read access • _SC_ARRAY_BOUND_ERROR_WRITE (= 503) Array boundaries exceeded during write access • _SC_VARIABLE_ACCESS_ERROR_READ (= 504) Error during read access: – To a system variable of a Technology Object: Violation of array limits; Access while Technology Object is being reset (RESET); Access to a Technology Object data type variable with a value of TO#NIL; – To an I/O variable by means of direct access. See Errors when accessing system variables and configuration data as well as I/O variables for direct access (Page 156). • _SC_VARIABLE_ACCESS_ERROR_WRITE (= 505) Error during write access: – To a system variable of a Technology Object: Violation of array limits; Access while Technology Object is being reset (RESET); Access to a Technology Object data type variable with a value of TO#NIL; – To an I/O variable by means of direct access. See Errors when accessing system variables and configuration data as well as I/O variables for direct access (Page 156). • _SC_TO_INSTANCE_NOT_EXISTENT (= 506) Access to a non-existent TO instance: A Technology Object data type variable with a value of TO#NIL is used in a TO function. • _SC_CLASS_INSTANCE_NOT_EXISTENT (= 507) Access to a non-existent class: An interface variable or pointer with value NULL is used. • _SC_TASK_STACKSIZE_FAULT (= 508) Stack size exceeded
TSI#taskId	:	StructTaskId	TaskId of task in which the execution error has occurred.
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: SYSTEM_INTERRUPT

5.3.6.8 TaskStartInfo of the PeripheralFaultTask

Table 5-32 TaskStartInfo of the PeripheralFaultTask

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#interruptID	:	UDINT	• _SC_PROCESS_INTERRUPT (= 200) Process alarm occurred at I/O module Additional information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#details – TSI#eventClass – TSI#faultId For information about process alarms, refer to **Process alarms** below. • _SC_DIAGNOSTIC_INTERRUPT (= 201) Diagnostic interrupt occurred at I/O module Additional information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr – TSI#details – TSI#eventClass – TSI#faultId • _SC_STATION_DISCONNECTED (= 202) PROFIBUS DP: A DP slave station has failed PROFINET IO: Station failure of an I/O device Additional information in the following TSI: – TSI#logDiagAdr – TSI#eventClass – TSI#faultId

Designation	:	Data type	Meaning
TSI#interruptID (continued)	:	UDINT	• _SC_STATION_RECONNECTED (= 203) PROFIBUS: Station reconnection of a DP slave PROFINET IO: Station reconnection of an I/O device Additional information in the following TSI: – TSI#logDiagAdr – TSI#eventClass – TSI#faultId • _SC_IMAGE_UPDATE_FAILED (= 204) Error during process image update (for DP slave: in connection with station failure) Additional information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#loDiagAdr See Errors when generating the process image (Page 158). • _SC_PC_INTERNAL_FAILURE (= 205) Error in local controller module Additional information in the following TSI: – TSI#details • _SC_IMAGE_UPDATE_OK (= 206) Process image update works again (for DP slave: In connection with station reconnection) Additional information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – #logDiagAdr See Errors when generating the process image (Page 158). • _SC_DP_CLOCK_DETECTED (= 207) Clock signal detected for the first time, and valid PRM telegram is received • _SC_DP_SYNCHRONIZATION_LOST (= 208) Multiple cycle failure or PLL has slipped (in internal DP_INTERFACES_SYNCHRONIZED state). PLL switches to uncontrolled mode • _SC_DP_SLAVE_SYNCHRONIZED (= 209) PLL has slipped into synchronized operation • _SC_DP_SLAVE_NOT_SYNCHRONIZED (= 210) Multiple cycle failure or PLL has slipped (in internal DP_SLAVE_SYNCHRONIZED state). PLL remains in controlled mode

Designation	:	Data type	Meaning
TSI#interruptID (continued)	:	UDINT	• _SC_IO_MODULE_SYNCHRONIZED (= 214) e.g. onboard SINAMICS measuring input (Control Unit) e.g. TM15 / TM17 High Feature / DO1 with telegram 39x: Synchronization attained. More information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) e.g. TM15/TM17 High Feature/DO1 with telegram 39x: Synchronization failed. More information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr • _SC_PULL_PLUG_INTERRUPT (= 216) PROFINET IO: Unplugging or plugging of modules in an I/O device. More information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#eventClass – TSI#faultId – TSI#logDiagAdr • _SC_DRIVE_OBJECT_FAULT (= 217) Fault message from DO1; cause of fault stored in the fault buffer, can be read out with _readDriveFaults. The alarm must be acknowledged with _resetDriveObjectFault. More information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr • _SC_DRIVE_OBJECT_ALARM (= 218) Warning message from DO1; warnings are stored in the warning parameters, can be read out with _readDriveParameter. The alarm cannot be acknowledged. It is triggered by the arrival and departure of the last warning (see TSI#details). More information in the following TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr – TSI#details

Designation	:	Data type	Meaning
TSI#logBaseAdrIn	:	DINT	Logic start address for following *TSI#interruptID* if the alarm was caused by an input range on the module: • _SC_PROCESS_INTERRUPT (= 200) • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) Otherwise _SC_INVALID_ADDRESS (= -1)
TSI#logBaseAdrOut	:	DINT	Logic start address for following *TSI#interruptID* if the alarm was caused by an output range on the module: • _SC_PROCESS_INTERRUPT (= 200) • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) Otherwise _SC_INVALID_ADDRESS (= -1)
TSI#logDiagAdr	:	DINT	Diagnostics address of a DP slave or I/O device for following *TSI#interruptID* • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_STATION_DISCONNECTED (= 202) • _SC_STATION_RECONNECTED (= 203) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_DRIVE_OBJECT_FAULT (= 217) • _SC_DRIVE_OBJECT_ALARM (= 218) Otherwise _SC_INVALID_ADDRESS (= -1)
TST#logDiagAdrIoType		DINT	Specifies the I/O type of a diagnostics address for the following *TSI#interruptID* • DSC_SVS_DEVICE_INPUT (= 198) • DSC_SVS_DEVICE_OUTPUT (= 199)
TSI#details	:	DWORD	Detailed information depending on the TSI#interruptID. See **Meaning of the TSI#details** table
TSI#eventClass	:	UINT	Event class depending on the TSI#interruptID Description in connection with TSI#faultId: See **Significance of the TSI#eventClass and TSI#faultId** table

Designation	:	Data type	Meaning
TSI#faultId	:	UINT	Fault ID depending on the TSI#interruptID Description in connection with TSI#eventClass: See **Significance of the TSI#eventClass and TSI#faultId** table
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: SYSTEM_:INTERRUPT

Table 5-33 Significance of TSI#details depending on TSI#interruptID (PeripheralFaultTask)

TSI#interruptID	Significance of TSI#details
_SC_PROCESS_INTERRUPT (= 200)	• Interrupt data of module signaling interrupt The structure of these data is presented in the module manual. • If interrupt source is in a SIMOTION I-slave: Call parameters of the *_sendProcessInterrupt()* system function
_SC_DRIVE_OBJECT_ALARM (= 218)	• 1 = Incoming warning • 0 = Outgoing warning
_SC_DIAGNOSTIC_INTERRUPT (= 201)	DS0 (= byte 0 - 3 of DS1) of module/station signaling interrupt The structure of the DS0 is described in the "Communication" function manual in the section "PROFINET IO - diagnostics and alarm behavior".

TSI#interruptID	Significance of TSI#details	
_SC_PC_INTERNAL_FAILURE (= 205)	The cause of the interrupt is represented as an OR operation to the following causes (sum of hexadecimal values).	
	Value	**Cause**
	16#00000001	"Fatal error" of host operating system (Windows blue screen - SIMOTION P).
	16#00000002	Temperature error (SIMOTION P, SIMOTION D)
	16#00000004	Temperature returned to normal (SIMOTION P, SIMOTION D)
	16#00000008	Battery warning (battery voltage low, but still sufficient - SIMOTION P, SIMOTION D)
	16#00000010	Battery voltage returned to normal (SIMOTION P, SIMOTION D)
	16#00000020	Battery failure (battery voltage too low – SIMOTION P, SIMOTION D)
	16#00000040	Battery module removed (SIMOTION P, SIMOTION D)
	16#00000080	Fan or dual fan failed (SIMOTION P, SIMOTION D)
	16#00000100	UPS in buffer state (SIMOTION P)
	16#00000200	UPS non-chargeable interval expired, PC is booting (SIMOTION P)
	16#00000400	UPS OK again (SIMOTION P)
	16#00000800	UPS battery warning (SIMOTION P)
	16#00001000	UPS battery error (battery no longer operational – SIMOTION P)
	16#00002000	UPS battery warning (SIMOTION P)
	16#00004000	One fan in dual fan failed (SIMOTION P, SIMOTION D)

Table 5-34 Significance of TSI#eventClass and TSI#faultId depending on TSI#interruptID (PeripheralFaultTask)

TSI#interruptID	TSI#eventClass	TSI#faultId	Bus system[1]	Meaning
_SC_PROCESS_INTERRUPT (= 200)	16#11	16#41	PROFIBUS DP PROFINET IO I/O bus	Process alarm
_SC_DIAGNOSTIC_INTERRUPT (= 201)	16#39[2]	16#42	PROFIBUS DP PROFINET IO I/O bus	Incoming diagnostic interrupt
	16#38[3]	16#42	PROFIBUS DP	Last error on the slot has gone
			PROFINET IO I/O bus	Last error on the subslot has gone

TSI#interruptID	TSI#eventClass	TSI#faultId	Bus system[1]	Meaning
_SC_STATION_DISCONNECTED (= 202)	16#39[2]	16#C4	PROFIBUS DP	A DP slave station has failed
		16#CA	PROFINET IO	System error PROFINET IO[4]
		16#CB	PROFINET IO	Station failure of a connected I/O device Shared I-device: No submodule of the I-device is currently subscribed through higher-level I/O controllers.
		16#CC	PROFINET IO	IO device fault present. Channel diagnostics or manufacturer-specific diagnostics pending. **Note:** 16#39:16#CB is always used for station failure and 16#38:16#CC is always used for the arrival with error
		16#F8	PROFINET IO	Partial station failure Shared iDevice: Submodules of the iDevice are partially subscribed to by the higher level IO controller.
_SC_STATION_RECONNECTED (= 203)	16#38[3]	16#C4	PROFIBUS DP	A DP slave station has been reconnected
		16#CB	PROFINET IO	Station reconnection of an IO device with error or warning. Shared iDevice: All submodules of the iDevice are currently subscribed to by the higher level IO controller.
		16#CC	PROFINET IO	Station reconnection of an IO device with error or warning.
		16#CD	PROFINET IO	An I/O device has been reconnected, but error: Set configuration <> actual configuration
		16#CE	PROFINET IO	An I/O device has been reconnected, but error in module parameterization
		16#F8	PROFINET IO	Partial station reconnection Shared iDevice: Submodules of the iDevice are partially subscribed to by the higher level IO controller.

TSI#interruptID	TSI#eventClass	TSI#faultId	Bus system[1]	Meaning
_SC_PULL_PLUG_INTERRUPT (= 216)	16#39[2]	16#51	PROFINET IO	PROFINET IO module has been removed or cannot be addressed.
		16#61	PROFIBUS DP	PROFINET DP module has been removed or cannot be addressed.
		16#54	PROFINET IO	PROFINET IO submodule has been removed or cannot be addressed.
		16#61	PROFIBUS DP	Station is faulted or module has been removed.
	16#38[3]	16#54	PROFINET IO	PROFINET IO module or submodule has been inserted, module type OK (actual configuration = set configuration)
		16#55	PROFINET IO	PROFINET IO module or submodule has been inserted, but wrong module type (actual configuration <> set configuration)
		16#56	PROFINET IO	PROFINET IO module or submodule has been inserted, but error during module parameterization
		16#58	PROFINET IO	I/O status of a module has changed from BAD to GOOD
		16#61	PROFIBUS DP	PROFINET DP module or submodule has been inserted, module type OK
		16#63	PROFIBUS DP	PROFINET DP module or submodule has been inserted, but wrong module type or module inserted

[1] Bus system which signals the TSI#interruptID with the specified TSI#eventClass and TSI#faultId

[2] Signals incoming event

[3] Signals outgoing event

[4] Outgoing event (TSI#eventClass = 16#38) is signaled for each existing station as station reconnection. Depending on the error status, the values 16#CB, 16#CD or 16#CE are displayed in TSI#faultId

Process alarms

Process alarms are indicated using the TSI "_SC_PROCESS_INTERRUPT".

Once a module triggers a process alarm, the alarm is read out, the PeripheralFaultTask is called and the alarm is acknowledged. If a process alarm of the same module occurs again during this time, note the following:

- If the process alarm occurs in the same channel that previously triggered a process alarm, then the relevant alarm is lost.
- If the process alarm occurs in a different channel of the same module, then no process alarm can be triggered at that moment. However, this alarm is not lost, but rather is triggered after the first active process alarm has been acknowledged. The behavior is analogous if a process alarm is triggered:
  - On another module of the same station
  - On another module of another station

5.3.6.9 TaskStartInfo of the TechnologicalFaultTask

Table 5-35 TaskStartInfo of the TechnologicalFaultTask

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#alarmNumber	:	DINT	Number of the triggered alarm (see description in the SIMOTION Alarms Diagnostics Manual) The parameters output in the alarm message are available in *TSI#alarmP1_DINT to TSI#alarmP5_LREAL* (e.g. *TSI#alarmP3_UDINT* is parameter 3 with data type UDINT).
TSI#toInst	:	ANYOBJECT	TO instance responsible for error, can be converted with the *AnyObject_to_Object* function. Comparison to the instance of the technology object is also possible, see example for using the TaskStartInfo of the TechnologicalFaultTask (Page 178).
TSI#commandId.low	:	UDINT	CommandId of triggering command (least significant word)
TSI#commandId.high	:	UDINT	CommandId of triggering command (most significant word)
TSI#alarmP1_DINT TSI#alarmP1_UDINT TSI#alarmP1_LREAL	: : :	DINT UDINT LREAL	Parameters 1 to 5 (associated value) of the message for the triggered alarm in the respective data type. Example: *TSI#alarmP3_UDINT* contains parameter 3 with UDINT data type. You can obtain the significance of the parameter from the description of the alarm in the SIMOTION Alarms Diagnostics Manual. This states the number and data type of the parameter with the following syntax: */n/%A* Where: • */n/* : Number of the parameter • *%A* : Abbreviation for the data type – *%d*: DINT – *%X*: UDINT – *%lf* : LREAL Example: */3/%X* means parameter 3 with UDINT data type. Only the parameters documented in the SIMOTION Alarms Diagnostics Manual for the triggered alarm are released for use. For information about process alarms, refer to *Process alarms* below.
TSI#alarmP2_DINT TSI#alarmP2_UDINT TSI#alarmP2_LREAL	: : :	DINT UDINT LREAL	
TSI#alarmP3_DINT TSI#alarmP3_UDINT TSI#alarmP3_LREAL	: : :	DINT UDINT LREAL	
TSI#alarmP4_DINT TSI#alarmP4_UDINT TSI#alarmP4_LREAL	: : :	DINT UDINT LREAL	
TSI#alarmP5_DINT TSI#alarmP5_UDINT TSI#alarmP5_LREAL	: : :	DINT UDINT LREAL	
TSI#AlarmType	:	ENUM...	Alarm type (as of version V4.5 of the SIMOTION kernel) TYPE EnumTechAlarmType FAULT WARNING INFO HIDDEN END_TYPE

Designation	:	Data type	Meaning
TSI#localReaction	:	ENUM...	Local action of the alarm (as of version V4.5 of the SIMOTION kernel) TYPE EnumTechAlarmLocalReaction NONE DECODE_STOP END_OF_MOTION_STOP MOTION_STOP MOTION_EMERGENCY_STOP MOTION_EMERGENCY_ABORT FEEDBACK_EMERGENCY_STOP OPEN_POSITION_CONTROL RELEASE_DISABLE OUTPUT_CAM_DISABLE MEASURING_INPUT_DISABLE FIXED_GEAR_DISABLE SIMULATION_STOP SIMULATION_ABORT ENCODER_DISABLE FOLLOWING_OBJECT_DISABLE DEPENDING_ON_OUTER_INPUT_LIMITCHECK_STATUS HEAT_OUTPUT_ZERO HEAT_AND_COOL_OUTPUT_ZERO DISABLE_MEASURE_AND_AVERAGE_OUTPUT_VALUE DISABLE_MEASURE_AND_MANUAL_OUTPUT_VALUE DISABLE_OUTPUT DISABLE_ALL CAMTRACK_DISABLE DISABLE CONTROL_STOP DISABLE_CONTROLLER FIXED_GEAR_DISABLE STOP_SPECIFIC_FORMULA STOP_ALL_FORMULA CONFIGURED_OUTPUT_VALUE END_TYPE
TSI#Alarmgroup	:	DINT	Alarm group (as of version V4.5 of the SIMOTION kernel)
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: SYSTEM_:INTERRUPT

5.3.6.10 TaskStartInfo of the TimeFaultBackgroundTask

Table 5-36 TaskStartInfo of the TimeFaultBackgroundTask

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:

Designation	:	Data type	Meaning
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#interruptID	:	UDINT	Triggering event: • _SC_BACKGROUND_TIMER_OVERFLOW (= 300) Additional events are not defined at this time.
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: SYSTEM_:INTERRUPT

5.3.6.11 TaskStartInfo of the TimeFaultTask

Table 5-37 TaskStartInfo of the TimeFaultTask

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#interruptID	:	UDINT	Triggering event: • _SC_CYCLE_TIMER_OVERFLOW (= 301) Additional events are not defined at this time.
TSI#taskId	:	StructTaskId	TaskId of TimerInterruptTask during which the time watchdog responded.
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: SYSTEM_:INTERRUPT

5.3.6.12 TaskStartInfo of the ShutdownTask

Table 5-38 TaskStartInfo of the ShutdownTask

Designation	:	Data type	Meaning
TSI#startTime	:	DT	Start time of the task
TSI#currentTaskId	:	StructTaskId	TaskId of task
TSI#cycleTime	:	TIME	Configured cycle time of task in ms (= 0, because the task is sequential)
TSI#cycleTime_us	:	UDINT	Configured cycle time of task in µs (= 0, because the task is sequential) SIMOTION Kernel as of version V4.4:
TSI#dwuser_1	:	DWORD	Reserved for internal use

Designation	:	Data type	Meaning
TSI#dwuser_2	:	DWORD	Reserved for internal use
TSI#shutDownInitiator	:	UDINT	Initiate transition to STOP: • _SC_MODE_SELECTOR (= 400): Mode selector switch • _SC_DEVICE_COMMAND (= 401): System function in user program • _SC_EXTERNAL_COMMAND (= 402): Command in SIMOTION SCOUT • _SC_EXCEPTION (= 403): Exception • _SC_ALARM_CONFIGURATION (= 404) Configured response to technological alarm Example of exceptions: • Execution errors in programs • Level overflow • Withdrawing a central module in RUN mode
TSI#taskType	:	ENUM...	Type of task As of version V4.5 of the SIMOTION Kernel Data type: EnumTaskType, see StartupTask (Page 161) Value: SHUTDOWN

5.3.6.13 Example for using the TaskStartInfo of the TechnologicalFaultTask

The following example shows you how to query the TO instance that initiated the alarm and the alarm number using the Taskstartinfo in the TechnologicalFaultTask. The *TO_AlarmProg* program must therefore be assigned to the TechnologicalFaultTask.

Table 5-39 Example of polling for Taskstartinfo in the TechnologicalFaultTask

```
PROGRAM TO_AlarmProg
    VAR
        dintVar  : DINT;
        dtVar    : DT;
    END_VAR;
    dtVar   := TSI#startTime;
    dintVar := TSI#alarmNumber;
    IF TSI#toInst = axis_1 THEN // Instance of the technology object
        ; // Statements
    END_IF;
    IF TSI#alarmNumber = 30002 THEN    // Triggering alarm
        ; // Statements
   END_IF;
END_PROGRAM
```

For an additional example, refer to Evaluating in the user program (Page 191).

Error Handling in Technology Objects 6

6.1 Possible errors in technology objects

The following basic errors are possible in the programming of technology objects:

- The technology object itself cannot execute the function required by the application or reports certain events or states:
  → A **technological alarm** is output.
  You can find information on the individual alarms in the SIMOTION Reference Lists.
- The command issued to a technology object cannot be executed:
  → The **return value** of the command provides information about the cause.
  You can find information on the return values of the commands in the SIMOTION Reference Lists.
- Error while accessing configuration data, system variables or I/O variables
  The ExecutionFaultTask is called in the event of errors when configuration data or variables are being read or written

See also

Process Alarms (Page 179)

Return values of commands (Page 192)

Errors when accessing system data with _get/_setSafeValue (Page 193)

6.2 Process Alarms

If an event (error, note) occurs on a technology object, the object issues a **technological alarm**.

A maximum of 160 technology object alarms can be stored in the system at one time. In this regard per TO identical alarms are counted as one alarm. If this buffer overflows because more than 160 alarms are pending simultaneously, the system switches to STOP mode with a **Message buffer overflow** diagnostics entry. The user cannot read out the buffer level.

If a series of the same alarms occur in succession, the detailed information will only be displayed on the first alarm. For the subsequent alarms the information will only be displayed when the first alarm that occurred has been acknowledged.

Effects of alarms

Technological alarms cause subsequent responses in the system. A distinction is made between:

- Effects on the affected technology object itself: **Local response**
- Effects on other technology objects or the execution system: **Global response**

For each alarm, certain effects have a default setting. However, you can adapt these settings to suit your requirements.

- By specifying the **error activation**, you can define whether the alarm is to be activated immediately, after repeated occurrences of an error, or after a certain period of time.
- You can hide some alarms. This allows you, for example, to suppress unimportant information.

Alarm group association

TO alarm numbers are assigned to an alarm group. Some alarm groups are defined for each TO type, whereas others are TO-specific. For example, alarm numbers 20006 and 20011 are assigned for a TO from the "Configuration error" alarm group.

The alarm group association of the pending technological alarms is displayed via the "errorGroup" system variable. A bit of this variable is assigned to each alarm group. Bit 1 (the lowest is bit 0) is assigned to the "Configuration error" alarm group. If bit 1 is set, this means there is an alarm from the "Configuration error" group at the TO. The following table lists the various alarm groups.

Description	Bit number	Meaning
SYSTEM_FAULT	0	System error
CONFIG_FAULT	1	Configuration error
USER_FAULT	2	User error
PERIPHERAL_FAULT	3	I/O error
FUNCTION_FAULT	4	Function/command error
FUNCTION_ABORTED	5	Function/command abort
RESET_RESTART_FAULT	6	Reset/restart error
DISTRIBUTED_MOTION_FAULT	7	Error during distributed synchronous operation
FOLLOWING_ERROR	8	Dynamic following error monitoring
STANDSTILL_POSITIONING_ERROR	9	Standstill/positioning monitoring error
DYNAMIC_LIMIT	10	Limitation of velocity, acceleration or jerk
CLAMPING_ERROR	11	Clamping monitoring error
SOFTWARE_LIMIT	12	Software limit position switch
LIMIT_SWITCH	13	Hardware limit position switch
SENSOR_FAULT	14	Encoder error
REFERENCE_NOT_FOUND	15	Homing output cam/zero mark error
OUTPUT_LIMIT	16	Output variable limitation
FORCE_DYNAMIC_LIMIT	17	Limitation of force and/or pressure
ADDITIONAL_SENSOR_FAULT	18	Error on additional encoder
SYNCHRONOUS_MOTION_FAULT	19	Synchronous operation error
FOLLOWING_OBJECT	20	Following object error
PATH_SYNCHRONOUS_MO-TION_FAULT	21	Synchronous path operation error
PATH_MOTION_FAULT	22	Path motion error
PATH_OBJECT	23	Path object error

See also

Local response (Page 181)
Global response (Page 181)
Error activation (Page 182)
Configure technological alarms (Page 183)
Displaying and acknowledging technological alarms (Page 185)
Acknowledging via the user program (Page 186)
Evaluating in the user program (Page 191)

6.2.1 Local response

With the *Local response* setting, you specify how the relevant technology object is to react when the alarm occurs and how subsequent commands for this TO will be handled.

Alarm responses are prioritized. Only one response can be in progress at any given time. This is the response with the highest priority of all the alarms pending at that particular point in time. When an alarm response occurs (except for NONE), the command decoder is always stopped. Any programmed commands issued subsequently are rejected. Command execution can continue after the alarm has been acknowledged in cases where the global error response for the alarm does not automatically require a Power ON.

You can select specific response options depending on the individual technology object and technology object alarm.

For information on the specific local response, please refer to the function manuals of the respective technology objects.

6.2.2 Global response

The *global response* describes the effect of a technology object alarm on the execution system. You can select the following response options depending on the respective TO alarm.

- **NONE**
  System does not respond when the alarm occurs.
- **START TechnologicalFaultTask**
  The **TechnologicalFaultTask** is started. Programs assigned to this task are started. This provides the user with the option of programming an application-specific response to the TO alarm. If there is no program assigned to this task, the system switches to STOP mode.

- **STOP**
  The system switches to **STOP** mode. In this mode, all technology objects are inactive, the user program is not executed, and all outputs are at zero. All system services are still active, and user programs can be loaded.
- **STOP U**
  The system switches to **STOP U** mode. The program execution is terminated, the axis enable and thus the position control is deactivated. The technology objects also remain active and can still process requests for testing and commissioning functions. Otherwise, identical to STOP mode.

6.2.3 Error activation

Type of error activation (TypeOfActivation)

Alarms and their associated alarm responses can be triggered in a variety of ways. Depending on the error type, the error activation is preset to one of the following values.

Table 6-1 Type of error activation

TypeOfActivation parameter	Meaning
ACTIVATE_IMMEDIATELY	The alarm is activated immediately when the error occurs.
ACTIVATE_AFTER_NTIME	The alarm is activated after the error occurs n times. The number n of error repetitions is set in the Continue parameter, see Error reproducibility table.
ACTIVATE_AFTER_NTIME_CONSECUTIVE	The alarm is activated after time t. The error must be pending without interruption over the entire time t. The time is set in the Time parameter, see Error response time table.

Error repeatability (Continue)

Table 6-2 Error repeatability

Continue parameter	Meaning
NO_TIME	The alarm is activated immediately when the error occurs.
TIME_n	The alarm is activated after the error occurs n times.

This parameter is relevant only if the **TypeOfActivation** parameter is set to ACTIVATE_AFTER_NTIME.
TIME_n can be set to values TIME_10, TIME_100, and TIME_1000, depending on the alarm.

Error response time (Time)

Table 6-3 Error response time

Time parameter	Meaning
NO_TIME	The alarm is activated immediately when the error occurs.
TIME_n	The alarm is triggered after a temporally uninterrupted error presence of n milliseconds.

This parameter is relevant only if the **TypeOfActivation** parameter is set to ACTIVATE_AFTER_NTIME_CONSECUTIVE.
TIME_n can be set to values TIME_1, TIME_10, TIME_100, and TIME_1000, depending on the alarm.

Type

You can set the **Type** parameter to specify whether certain technology object alarms are to be displayed. If you select the *Hidden* setting, an alarm message is not displayed when this alarm occurs and no entry is written to the diagnostics buffer. You can thereby prevent an overflow of the alarm buffer when a particular technology alarm is issued frequently, or suppress a note message that is not important to you.

6.2.4 Configure technological alarms

Individual responses are preset for each alarm. To change these default settings, proceed as follows:

1. Select the **Execution System** path in the project navigator. The **Execution System** window opens. Select the path **SystemInterruptTasks → TechnologicalFaultTask**. Then click the **Alarm configuration** button in the window.

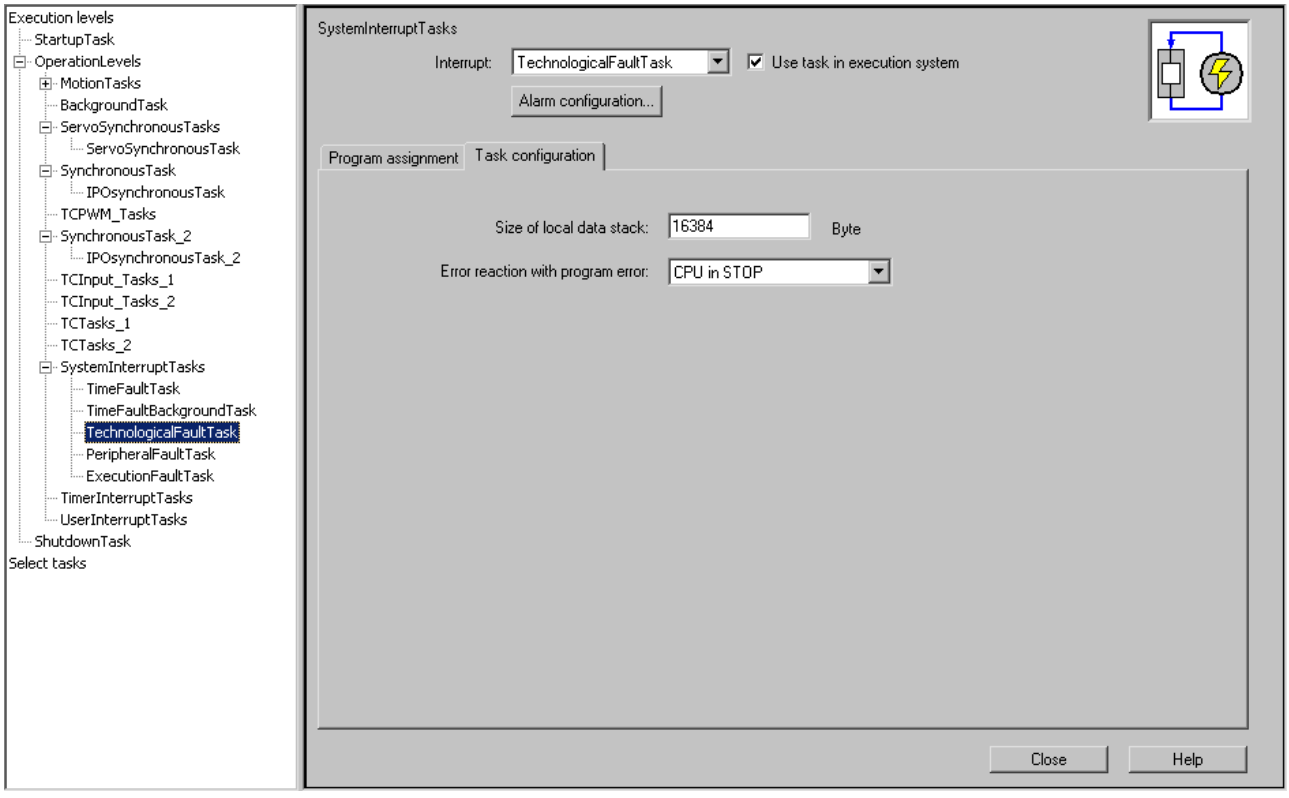


Figure 6-1 Configuring a technological alarm

2. In the combo box, select the technology object for which you want to configure alarms. The alarms for the technology object are displayed.

3. Select the alarm for which you want to change the response.
4. Select the required response from the list for the corresponding technology object alarm. The options available depend on the type of alarm.

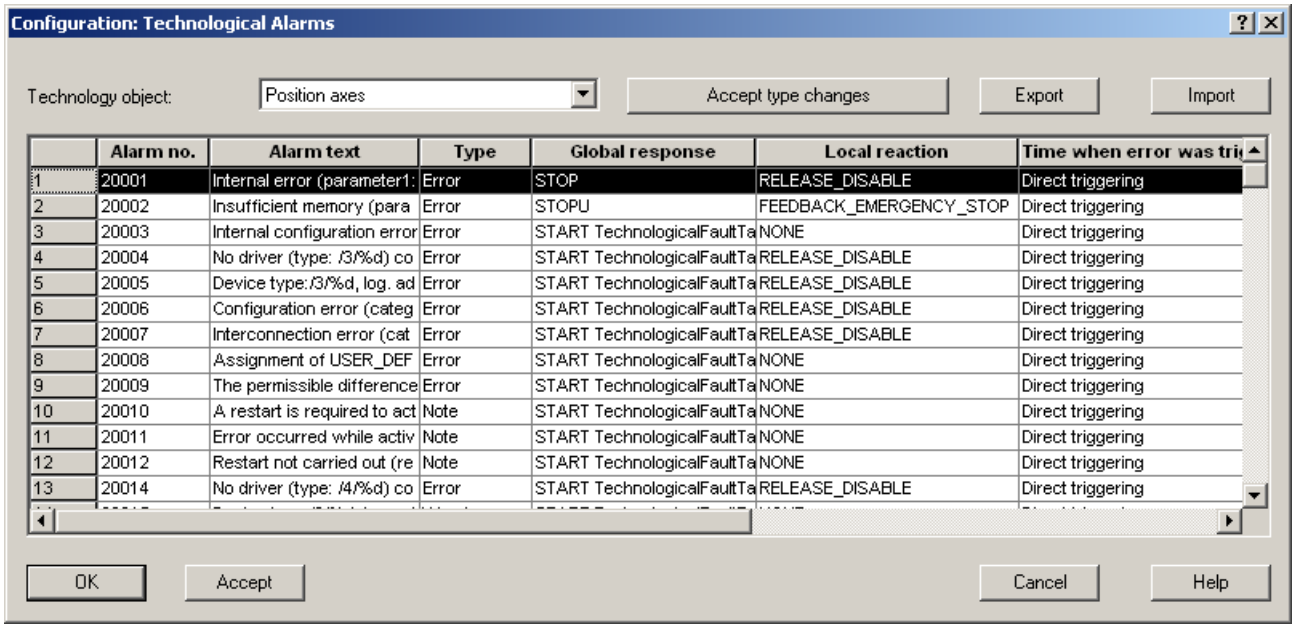


Figure 6-2 Selecting a technological alarm

Note

The technology object whose alarms you want to configure must already be configured.

The alarm configuration can be transferred to other technology objects via the corresponding buttons in the **Alarm configuration** dialog (via export/import).

Exporting or importing technological alarms

Note

So that all changes that you have made in the dialog **Configuration: Technological alarms** are exported, the button **Export** remains inactive until you have applied all changes using **Apply**.

To export all TO alarms in XML format, proceed as follows:

1. Open the **Configuration: Technological alarms** dialog.
2. Click on export to select the **Export alarm configuration** dialog.
3. Select a path for the export and confirm with **OK**.
   The alarms are stored there.

To import TO alarms in XML format, proceed as follows:

1. In the **Configuration: Technological alarms** dialog click on Import.
   This displays the **Import alarm configuration** dialog.
2. Under **Source path and source name of the import** select the desired XML file.
3. Click **OK** to import the data.
   The alarms will then be displayed in the dialog.

Configuring messages (alarms) for all TOs of a certain TO type

If several technology objects of the same TO type are configured on the SIMOTION device, you can assign one alarm configuration to all TOs of this type. For example you can assign a configuration to all position axes.

1. Open the **Configuration: Technological alarms** dialog.
2. Select the TO type from the list, e.g. **- All positioning axes**.
3. Click on **OK** to assign the settings to all preciously configured TOs of the selected type.

6.2.5 Displaying and acknowledging technological alarms

Technological alarms can be evaluated and acknowledged in different ways:

- When SIMOTION SCOUT is in **online mode**, alarms and messages are displayed on the **Alarms** tab in the detail view of the workbench.
  With **Acknowledge**, all alarms of the associated type are deleted.
- Alarms can be output, displayed, and acknowledged via the **H**uman **M**achine **I**nterface (HMI).
- All pending or individually selected alarms of a technology object can also be queried, evaluated, and acknowledged via the user program.

Note

If during the acknowledgement of TO alarms, a new alarm occurs at the TO with the same or lower priority, then the new alarm is no longer reported in the system. This behavior must be taken into account during error acknowledgement.

Acknowledging via SIMOTION SCOUT

Acknowledging individual alarm:

1. Select the alarm in the **Alarms** tab of the detail view.
2. Click **Acknowledge**.

Acknowledging all alarms:

- Click on **Acknowledge all**.

All alarms of the associated type are then deleted.

Note

Because drive alarms usually generate technology object alarms as well, the drive alarms are also deleted with the **Acknowledge (TO)** switch. If however the cause of a drive alarm still exists then a new TO alarm will be triggered immediately. In this case first correct the cause of the drive alarm.

Display and acknowledge via HMI

1. Connecting via WinCC flexible
   Alarms are displayed in a configured message line or in message windows.
   WinCC devices are acknowledged via the ACK key or via user-configured softkeys or buttons.
   (For operating instructions, see **WinCC flexible** description)
2. Link via OPC
   Alarms can be displayed and acknowledged in SimaticNet as of V6.0 SP4 OPC Alarms & Events. In addition the diagnostic buffer can be read out and acknowledged if necessary.
   (For handling, see the *SIMOTION IT Diagnostics and Configuration* Function Manual which you can find on the SIMOTION SCOUT CD documentation)

6.2.6 Acknowledging via the user program

Acknowledge all pending technology object alarms

_resetTechnologicalErrors → Acknowledge all currently pending technology object alarms

ST call example: Acknowledge all pending alarms

```
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END_INTERFACE

IMPLEMENTATION
    PROGRAM EXAMPLE
        VAR
            s_i_RetVal : DINT;
        END_VAR;
        (* Acknowledge all TO alarms *)
        s_i_RetVal := _resetTechnologicalErrors();
    END_PROGRAM
END_IMPLEMENTATION
```

MCC call: Acknowledge all pending alarms

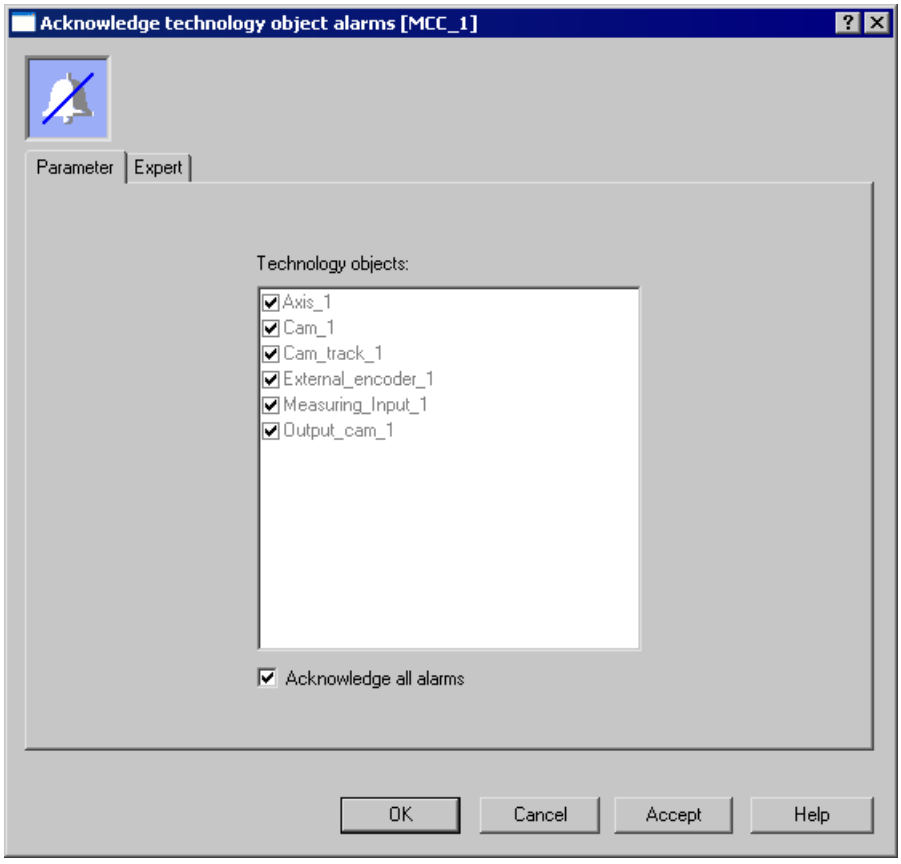


Figure 6-3 MCC call: Acknowledge all pending alarms

Acknowledge all pending alarms of a technology object

ST call example: Acknowledge all alarms on a TO axis, TO measuring input and TO output cam

```
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END_INTERFACE

IMPLEMENTATION
    PROGRAM EXAMPLE
        VAR
            s_i_RetVal : DINT;
        END_VAR;
        (* Acknowledge TO alarms ('ResetTOAlarms') *)
        s_i_RetVal := _resetAxisError(axis := Axis_1);
        s_i_RetVal := _resetMeasuringInputError(
                measuringInput := Measuring_input_1);
        s_i_RetVal := _resetOutputCamError(
                outputCam := Output_cam_1);
    END_PROGRAM
END_IMPLEMENTATION
```

MCC call: Acknowledge all alarms on a TO axis, TO measuring input and TO output cam

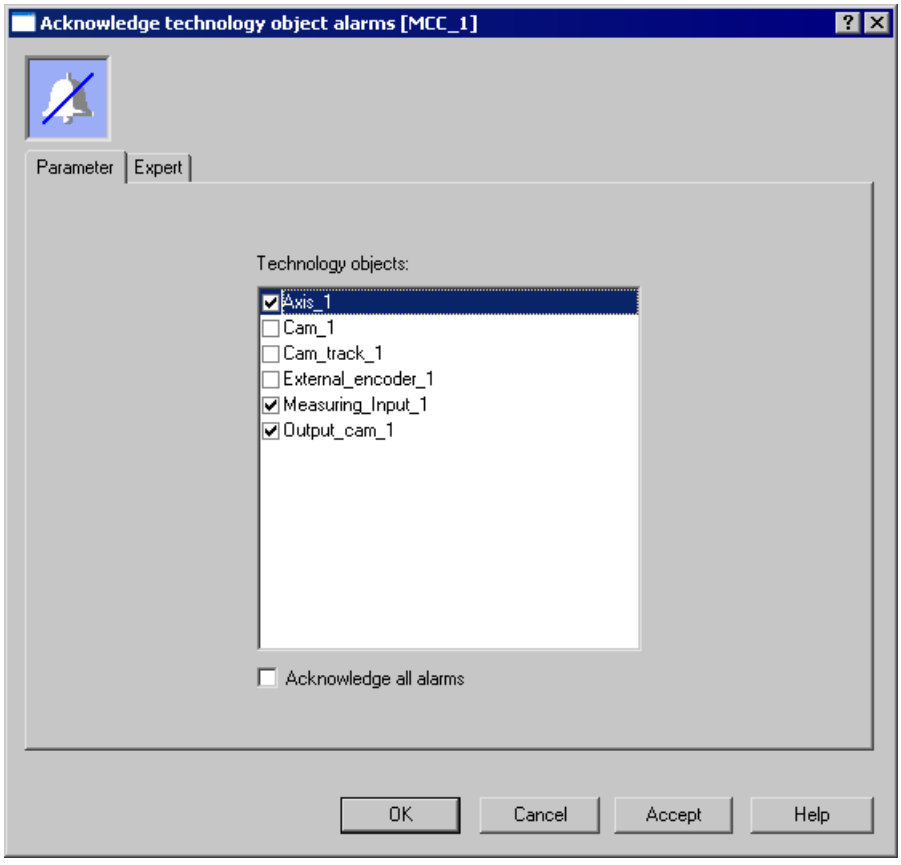


Figure 6-4 MCC call: Acknowledge all alarms on a TO axis, TO measuring input and TO output cam

Acknowledging a specific alarm of a technology object

By adding **errorResetMode** := SPECIFIC_ERROR and **errorNumber** := XXXXX to the **_reset..** commands, it is possible to specifically acknowledge a certain alarm.

Example:

_resetAxisError (..., **errorResetMode** := SPECIFIC_ERROR, **errorNumber** := 30002)

ST call example: Acknowledge alarm 30002 on a TO axis

```
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END_INTERFACE

IMPLEMENTATION
    PROGRAM EXAMPLE
        VAR
            s_i_RetVal : DINT;
        END_VAR;
        (* Acknowledge specific TO alarm ('ResetSingleTOAlarm') *)
        s_i_RetVal:= _resetAxisError(axis := Axis_1,
                errorResetMode:= SPECIFIC_ERROR,
                errorNumber := 30002);
    END_PROGRAM
END_IMPLEMENTATION
```

MCC call: Acknowledge alarm 30002 on a TO axis

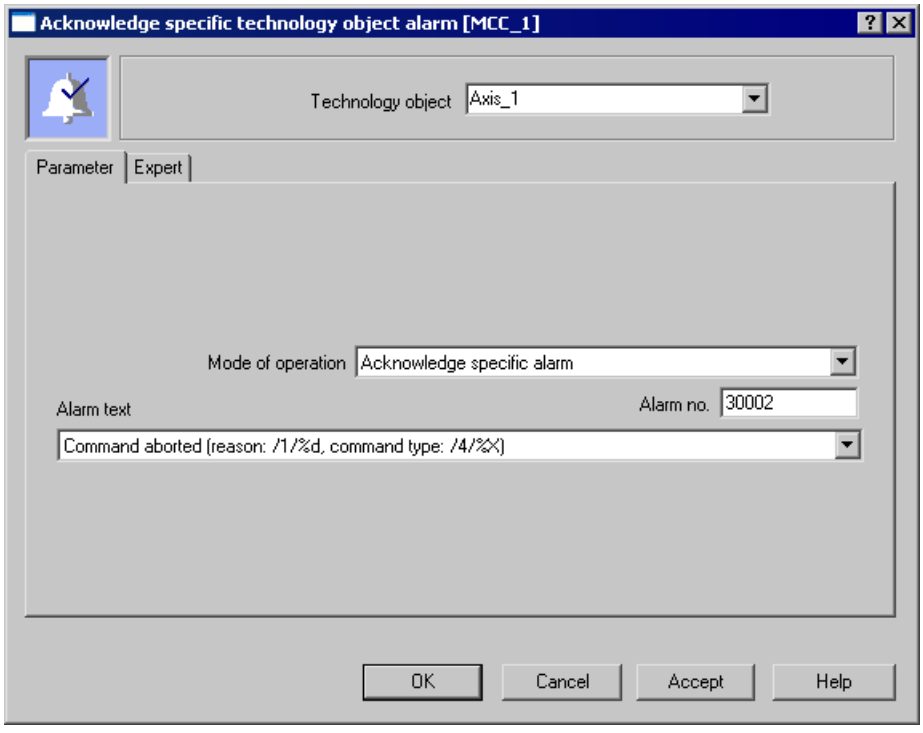


Figure 6-5 MCC call: Acknowledge alarm 30002 on a TO axis

Resetting a technology object

The technology object is set to its initial state and all pending alarms are acknowledged.

Note

In addition to troubleshooting the commands for reset have other effects on the TO as well. Therefore, as a general rule, errors should be acknowledged with the **_reset...Error** commands.

In contrast to the **_reset...Error** command, the **_reset...** commands also bring the respective TO to a safe state. In addition to acknowledging the alarms, the following actions are performed for each TO type:

- Stop the active commands
- Clear the command buffer
- Reset system variables (parameter userDefaultData)
- Execute a TO restart (parameter activateRestart)

In addition, actions dependent on the TO type are executed:

- Axes: Generate a braking ramp
- Output cam and cam track: Switch off the hardware output
- Cam: Delete the curve geometry
- Path object: Stop the path group

Call example: Resetting TO axis, TO measuring input and TO output cam

```
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END_INTERFACE

IMPLEMENTATION
    PROGRAM EXAMPLE
        VAR
            s_i_RetVal : DINT;
        END_VAR;
        (* Reset object ('ResetObject') *)
        s_i_RetVal := _resetAxis(
                axis := Axis_1,
                userDefaultData := DO_NOT_CHANGE);
        s_i_RetVal := _resetMeasuringInput(
                measuringInput := Measuring_input_1,
                userDefaultData := DO_NOT_CHANGE);
        s_i_RetVal := _resetOutputCam(
                outputCam := Output_cam_1,
                userDefaultData := DO_NOT_CHANGE);
    END_PROGRAM
END_IMPLEMENTATION
```

6.2.7 Evaluating in the user program

In the case of alarms for which the **StartTechnologicalFaultTask** global response is configured, the **TechnologicalFaultTask** is called once with every alarm occurrence. The number of the pending alarm and the technology object triggering the alarm can be scanned in this task. The information is passed to the **TechnologicalFaultTask** via the task start info (TSI).

You can add a program to the **TechnologicalFaultTask** and thus program an individual error response for specific alarms or, for example intercept alarms and forward them to higher-level alarm evaluation.

Note

If you have not added any program to the **TechnologicalFaultTask**, the CPU will switch to **STOP** mode if the task is called by an alarm.

The following parameters are transferred in **TechnologicalFaultTask** for evaluation:

TSI#startTime	→	time at which the alarm was registered
TSI#alarmNumber	→	alarm number
TSI#toInst	→	name of the technology object that triggered the alarm (e.g. Axis_1)

Programming example

Every time **Alarm 30002** occurs on Axis_1, a counter (**s_i_Count**) is to be incremented and the alarm acknowledged automatically.

Note: This sample program must be added to the **TechnologicalFaultTask** in the execution system!

```
UNIT ST_1;
INTERFACE
  USEPACKAGE CAM;
  PROGRAM TO_AlarmProg;
END_INTERFACE

IMPLEMENTATION
  PROGRAM TO_AlarmProg
    VAR
      s_i_Count : INT;
      s_i_RetVal: DINT;
    END_VAR;

    (* Query whether alarm 30002 is pending for Axis_1 *)
    IF (TSI#alarmNumber = 30002) AND
                        (TSI#toInst = Achse_1) THEN
      (* Increment counter *)
      s_i_Count:= s_i_Count + 1;
      (* Acknowledge specific technology object alarm *)
      s_i_RetVal := _resetAxisError (
                        axis:=Achse_1,
                        errorResetMode := SPECIFIC_ERROR,
                        errorNumber := 30002);
    END_IF;
  END_PROGRAM
END_IMPLEMENTATION
```

6.3 Return values of commands

The user program indicates whether or not an executed command was able to be executed successfully during runtime.

Any error information is contained in the return value of the block. You do not have to acknowledge this error information.

Evaluating the return value

When a command returns a value of zero, this signifies that the command has been executed without error. If an error has occurred, the return value contains an error number that can be used to identify the cause of the error.

The possible error numbers for the respective commands can be found in the SIMOTION Reference Lists.

You can respond to possible errors during system function execution in the user program, thus preventing secondary errors.

Likewise for MCC commands you can program a specific error response by entering a return variable with the return value of the command (see MCC Programming Manual, section Expert tab).

Call example

If an error occurs when the **_pos** command is being executed, the **g_bo_error** variable is to be set to **true**, the return value entered in the **g_i_errornumber** parameter and the block aborted.

Note: This sample program must be added to a **MotionTask** in the execution system!

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
        VAR_GLOBAL
            g_bo_error: BOOL;
            g_i_errornumber: DINT;
            g_i_RetVal: DINT;
        END_VAR
    PROGRAM RETURN_VALUE;
END_INTERFACE
IMPLEMENTATION
    PROGRAM RETURN_VALUE

         (* Position axis ('Pos') *)
        g_i_RetVal:= _pos(axis:=Axis_1,
        direction:=SHORTEST_WAY,
        positioningMode:=ABSOLUTE,
        position:=222,
        velocityType:=DIRECT,
        velocity:=1000,
        velocityProfile:=TRAPEZOIDAL,
        blendingMode:=INACTIVE,
        mergeMode:=IMMEDIATELY,
        nextCommand:=WHEN_MOTION_DONE,
        commandId:=_getCommandId());
         (*Evaluation of the return value *)
        IF g_i_RetVal <> 0 THEN
            g_i_errornumber := g_i_RetVal;
            g_bo_error := true;
            Return;  // End block
        END_IF;
    // Further user program as of here.
    END_PROGRAM
END_IMPLEMENTATION
```

6.4 Errors when accessing system data with _get/_setSafeValue

Error while accessing configuration data, system variables or I/O variables

With V4.1.3 and higher, these system functions are not necessarily required for system variables and config data. In the event of access errors (e.g. technology object in restart), a "replacement value" or "last value" can be configured; see System variables (Page 143) or Configuration data (Page 147).

Where direct access is involved, ExecutionFaultTask is called in the event of errors occurring when configuration data or system variables are being read or written (see Access errors to system variables and configuration data, as well as I/O variables for direct access (Page 156)).

However, in certain cases it is necessary to avoid calling the ExecutionFaultTask, to respond differently to an error or to deviate from the configured error response. The functions _getSafeValue, _setSafeValue, and getInOutByte serve this purpose. Configuration data item *restartInfo.behaviorInvalidSysvarAccess* can be used for accessMode = CONFIGURED (see System variables (Page 143) or Configuration data (Page 147)).

Replacement value strategy and errors when accessing system variables, configuration data, and I/O variables

There are various situations in which the reading or writing of a system variable, a configuration data item or an I/O variable can fail.

Table 6-4 Possible causes for the read/write failure

Cause	System variable	Config data item	I/O variable
Variable temporarily not available	TO in restart TO deactivated	TO in restart TO deactivated	Faulty input
	-	Data not visible in the current configuration	Faulty output
Illegal value (for write)	Value lies outside the limits	Value lies outside the limits	-
	Value not in the grid	Value not in the grid	-

Depending on the cause, the following responses are possible:

Table 6-5 System response in the event of an error when using _getSafeValue

Required response		System variable	Config data item	I/O variable
A value that temporarily cannot be accessed while reading:				
Go to STOP	Call = accessMode	STOP_DEVICE	STOP_DEVICE	STOP_DEVICE
	Answer	→ STOP	→ STOP	→ STOP
	Value	Unchanged	Unchanged	Unchanged
Do not read	Call = accessMode	NO_CHANGE	NO_CHANGE	Not implemented
	Answer	NO_CHANGE	NO_ACCESS	
	Value	Last value	Configured value	
Read substitute value	Call = accessMode	DEFAULT_VALUE	DEFAULT_VALUE	DEFAULT_VALUE
	Answer	NO_ACCESS	INVALID_VALUE	DEFAULT_VALUE
	Value	Default value	Default value	Substitute value
Read last valid value	Call = accessMode	CONFIGURED	CONFIGURED	NO_CHANGE
	Answer	(In acc. with configuration data item behavior, etc.)	(In acc. with configuration data item behavior, etc.)	NO_CHANGE
	Value			Last valid value

Table 6-6 System response in the event of an error when using _setSafeValue - when writing a temporarily inaccessible value

Required response		**System variable**	**Config data item**	**I/O variable**
Go to STOP	Call = accessMode	STOP_DEVICE	STOP_DEVICE	STOP_DEVICE
	Answer	→ STOP	→ STOP	→ STOP
	Value	Unchanged	Unchanged	Current value
	Return value setValue	Current value	Current value	-
Do not write	Call = accessMode	NO_CHANGE	NO_CHANGE	Not implemented
	Answer	NO_CHANGE	NO_ACCESS	
	Value	Unchanged	Unchanged	
	Return value setValue	Current value	Current value	
Do not write	Call = accessMode	DEFAULT_VALUE	DEFAULT_VALUE	Not implemented
	Answer	NO_ACCESS	NO_ACCESS	
	Value	Unchanged	Unchanged	
	Return value setValue	Current value	Current value	
Do not write	Call = accessMode	CONFIGURED	CONFIGURED	Not implemented
	Answer	(In acc. with configuration data item behavior, etc.)	(In acc. with configuration data item behavior, etc.)	
	Value			
	Return value setValue	Current value	Current value	
Accept substitute value, will take effect later	Call = accessMode	Not implemented	Not implemented	DEFAULT_VALUE
	Answer			DEFAULT_VALUE
	Value			Substitute value
Accept current value, will take effect later	Call = accessMode	Not implemented	Not implemented	NO_CHANGE
	Answer			NO_CHANGE
	Value			Current value

Table 6-7 System response in the event of an error when using _setSafeValue - error when writing an invalid value

Required response		System variable	Config data item	I/O variable
Go to STOP	Call = accessMode	STOP_DEVICE	STOP_DEVICE	Does not occur
	Answer	→ STOP	→ STOP	
	Value	Unchanged	Unchanged	
	Return value setValue	Current value	Current value	
Do not write	Call = accessMode	NO_CHANGE	NO_CHANGE	
	Answer	NO_CHANGE	NO_CHANGE	
	Value	Unchanged	Unchanged	
	Return value setValue	Current value	Current value	
Write default value	Call = accessMode	DEFAULT_VALUE	DEFAULT_VALUE	
	Answer	DEFAULT_VALUE	INVALID_VALUE	
	Value	DEFAULT_VALUE	Unchanged	
	Return value setValue	Current value	Current value	

Legend for tables above

- DEFAULT_VALUE
  The DEFAULT_VALUE is the configured value. This is the value transferred during a download (system variables and config data).
- LAST VALUE - configured value (config data)
  The last value can be the configured value. The value will then be equivalent to the value shown under "Current value" in the expert list.
  OR
  The last value can be the last configured value. The value will then be equivalent to the value shown under "Next value" in the expert list.
  Either of these values can be output, depending on the configuration data entered.
- LAST VALUE - last value (system variables)
  With system variables, the last value is the value which is currently set and effective.

See also

System variables (Page 143)
_getSafeValue function (Page 420)
_setSafeValue function (Page 422)
_getInOutByte function (Page 426)
Configuration data (Page 147)

Execution System, Tasks, and System Cycle Clocks 7

7.1 Execution system

The SIMOTION **execution system** provides various **execution levels**.

- The execution levels are assigned to **user program tasks**.
  - **Programs** are assigned to the user program tasks.

  All programs - and thus also tasks - can contain PLC and motion control tasks.

The following execution levels are available:

- Synchronous execution levels: synchronous with control and interpolator cycle clocks
- Time-driven execution levels
- Event-driven execution levels
- Interrupt-controlled execution levels
- Sequential execution levels
- Free-running execution levels

Various **tasks** with different execution properties are available for specific tasks:

System tasks

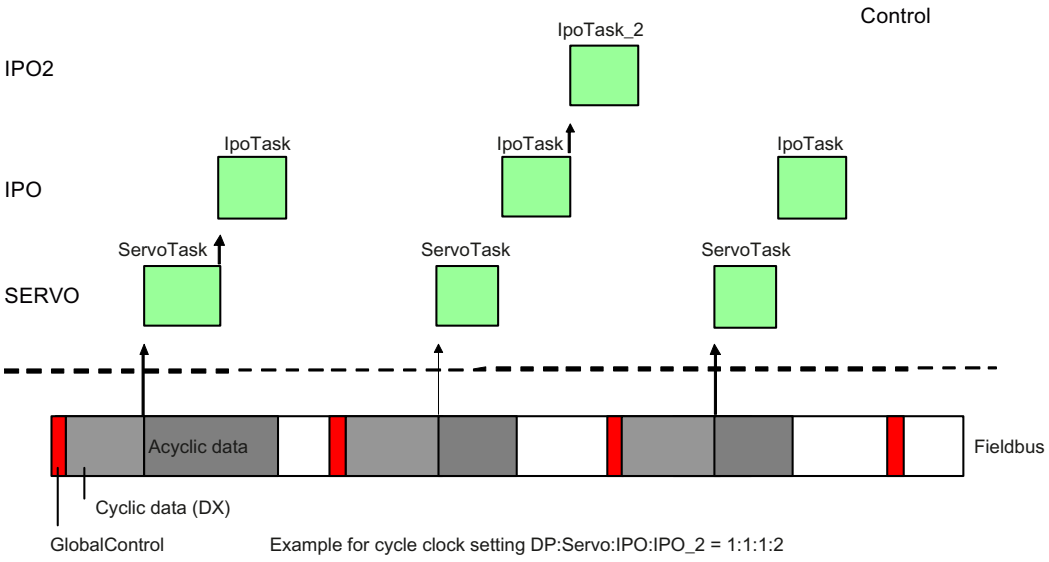
System tasks are regularly executed by the system.

The system cycle clock can be specified.

The following tasks are executed by system tasks:

- **Communication**
  System tasks for the connection to the isochronous PROFIBUS, to PROFINET IO with IRT or the system cycle clock and for I/O processing.
- **Motion control**
  - Base cycle clock/bus cycle clock
    The base/bus cycle clock (DP cycle clock or PN cycle clock) is based on the cycle of the isochronous bus and determines the intervals for data exchange with the DP/PN I/O. The base/bus cycle clock is used as the basis for setting further cycle clocks.
  - Position control cycle clock
    Among other things, position control and monitoring of axes, drive communication, and I/O processing take place during the time slice of the position control cycle clock. The cycle for the position control cycle clock determines the interval between two position controller cycle clocks. The position control cycle clock can be set as a multiple of the base/bus cycle clock. A cycle clock ratio of 1:1 is recommended.
  - IPO cycle clock
    The time slice of the interpolator cycle clock is used, amongst other things, to calculate the setpoints. The cycle for the interpolator cycle clock determines the interval between two interpolator cycle clocks. Time slice requirements may increase briefly at the start of one or more commands. The IPO cycle clock can be set as a multiple of the position control cycle clock. A cycle clock ratio of 1:1 is recommended.
  - IPO2 - cycle clock 2
    Interpolator cycle clock 2 handles the same tasks as the interpolator cycle clock and can be used for technology objects of lower priority classes. The IPO2 cycle clock can be set as a multiple of the IPO cycle clock.
  - Servo_fast/IPO_fast
    An additional faster position control cycle clock option and an IPO option can be configured as of V4.2, see also Servo_fast (application cycle clock for fast bus system) (Page 264).

  Adjustable gear ratios between bus task, servo, and IPO support appropriate load distribution and optimum system utilization.
  With digital drives, these execution levels are synchronized with the isochronous PROFIBUS or PROFINET IO with IRT. For analog drives without isochronous PROFIBUS, the execution levels are supplied with an adjustable system cycle clock from an internal timer.



- **Temperature control**
  In connection with the **TControl** technology package, execution levels are available for the temperature control: Actual value measurement, control and pulse width modulation of the output signals.

User programming

Execution levels for the task-related programming are available for the user programming (**user program tasks**): Motion control, logic and technological functions.

See also

Specifications for the configuring (Page 320)

7.1.1 Execution levels / tasks

The execution levels define the chronological sequence of programs in the execution system. Each execution level contains one or more tasks.

A task provides the execution framework for the programs. Each task is executed when specific conditions are satisfied. You can assign one or more user programs to each task and specify their order within the task.

Besides user program tasks there are also several system tasks, the contents and execution sequence of which you cannot influence.

Execution levels

The following figure shows the execution levels with their tasks.

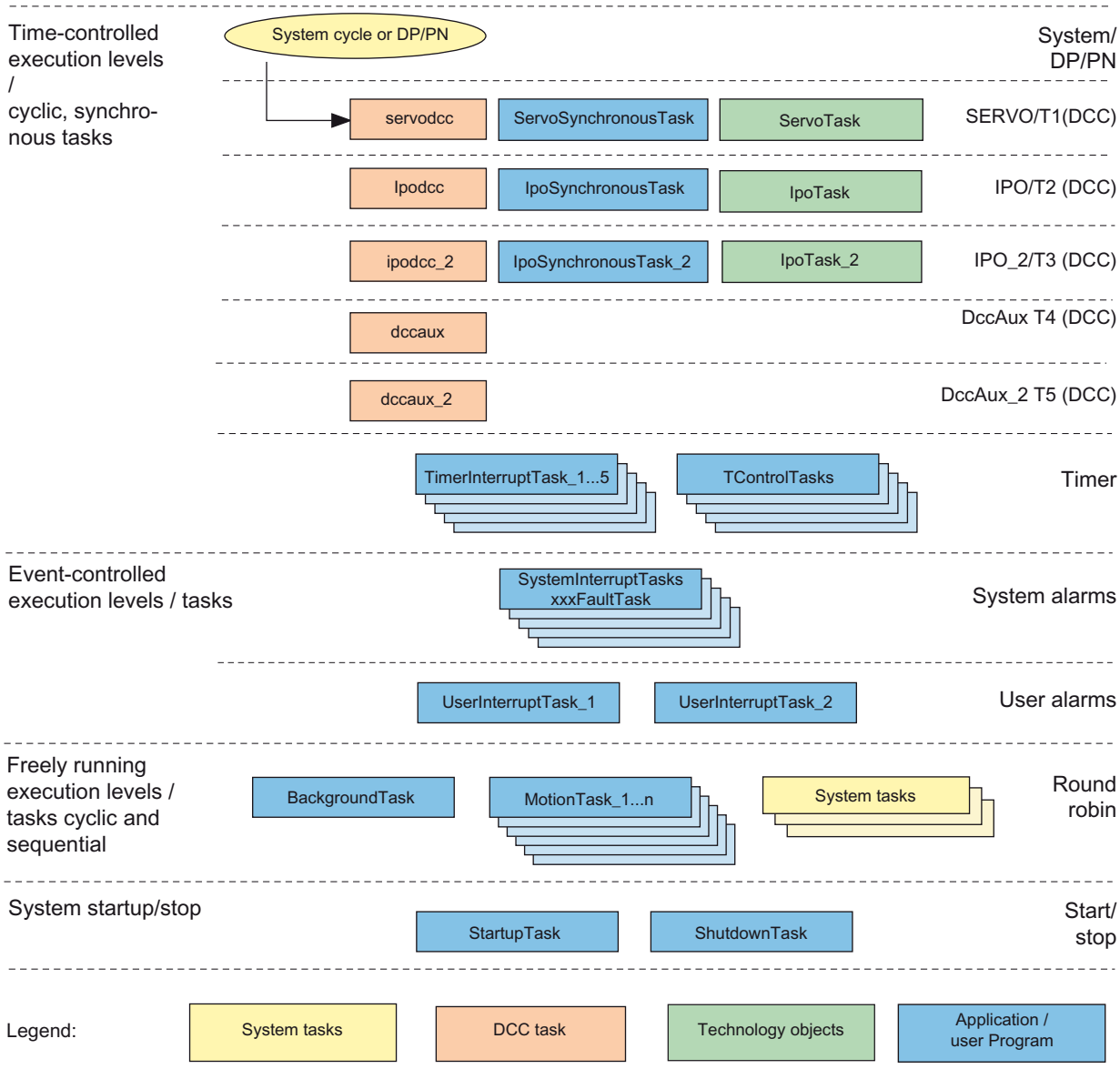


Figure 7-1 Execution levels and tasks in the SIMOTION runtime system

When you use technology packages, the execution levels provided by the system are automatically assigned. The user program cannot influence the processing of technology functions in these execution levels. System tasks are acyclic communication (Ethernet, PROFIBUS DP, PROFINET IO), debug services or trace preprocessing, for example.

The DCC levels and tasks are not visible in the execution system in the SCOUT user interface. The DCC plans are arranged in the DCC editor via execution groups, see **Description of the DCC editor**.

The following example for the Axis technology object demonstrates how the system and user program tasks interact with each other. At the end of cyclic data transmission, the ServoSynchronousTask and ServoTask are started, followed by the IPOSynchronousTask and IPOTask.

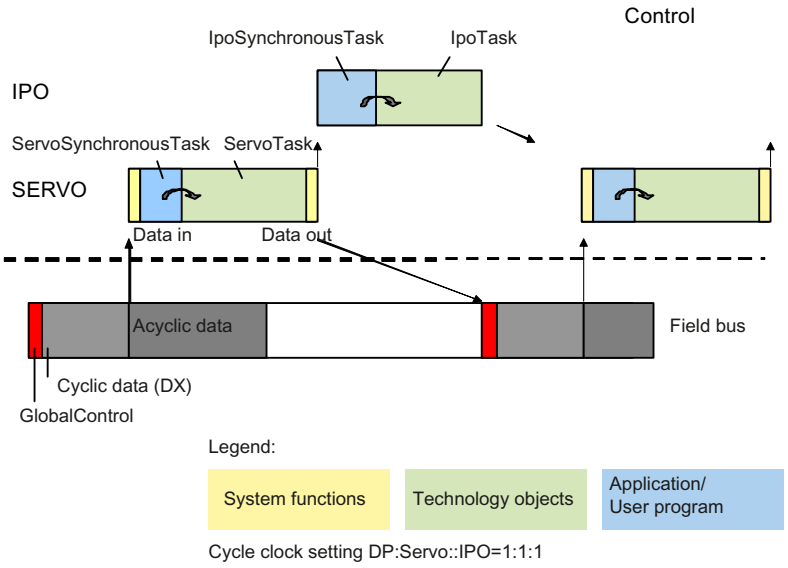


Figure 7-2 Task interaction

For more information, go to:

Isochronous I/O processing on fieldbus systems (Page 283)

Dynamic response with respect to data processing in the control (Page 289)

Tasks

One or more tasks are available in each execution level for user programming.

The main features of the tasks are:

- Start behavior: When and under what conditions a task is started. (refer to Table)
- Priority: Which task is interrupted by which other task. (refer to the table for the task priorities in the next section)

The following table shows the tasks available for user programs.

Table 7-1 Tasks in the SIMOTION runtime system

Task		Description
StartupTask		The **StartupTask** is executed once at the transition from **STOP** or **STOP U** mode to **RUN** mode. It is intended for initialization and resetting of technology objects.
Free-running tasks:		In the **round robin execution level**, the **MotionTasks** and **BackgroundTask** are executed by the system in the background in the time-slice procedure.
	MotionTasks	**MotionTasks** are intended for the programming of sequences, for programmed motion control or other *sequential* executions. MotionTasks are started by user programs and executed once.
	BackgroundTask	The **BackgroundTask** is provided for the programming of *cyclic* sequences without a fixed time frame. The BackgroundTask is started after system start-up and then executed cyclically and free-running.
Time-driven tasks and synchronous tasks:		**Cyclic tasks**. They are called cyclically in a certain time frame and are automatically restarted after the execution of the assigned programs.

Task		Description
	TimerInterruptTasks	**TimerInterruptTasks** are intended for the periodical starting of programs.
	SynchronousTasks	**SynchronousTasks** are started periodically, synchronous with a specified system cycle clock.
Event-driven tasks:		**Sequential tasks**. They are started and executed once when an event occurs and then terminated.
	SystemInterruptTasks	**SystemInterruptTasks** are started and executed once when a system event occurs.
	UserInterruptTasks	**UserInterruptTasks** are started and executed once when a user-defined event occurs.
ShutdownTask		The **ShutdownTask** is executed once at the transition from **RUN** mode to **STOP** or **STOP U** mode.

Note

In addition to the user program tasks, there are various system-internal tasks that the user cannot influence.

The **ControlPanelTask** is such a system-internal task, for example. It is visible during the task run times of the device diagnosis, but not in the execution system.

As of V4.1.2, the TaskTrace is also available. The SIMOTION Task Trace records the sequence of the individual tasks, labels "user events", which you can generate using a program command, and represents these graphically; see the Task Trace Function Manual.

See also

StartupTask (Page 212)

MotionTasks (Page 215)

BackgroundTask (Page 218)

TimerInterruptTasks (Page 221)

SynchronousTasks (Page 224)

SystemInterruptTasks (Page 232)

UserInterruptTasks (Page 237)

ShutdownTask (Page 242)

Isochronous I/O processing on fieldbus systems (Page 283)

7.1.2 Execution system in SIMOTION SCOUT

All tasks used in the execution system can be viewed in SIMOTION SCOUT.

- Select the SIMOTION device in the project navigator and select **Target system > Configure execution system** in the menu or double-click **EXECUTION SYSTEM**.

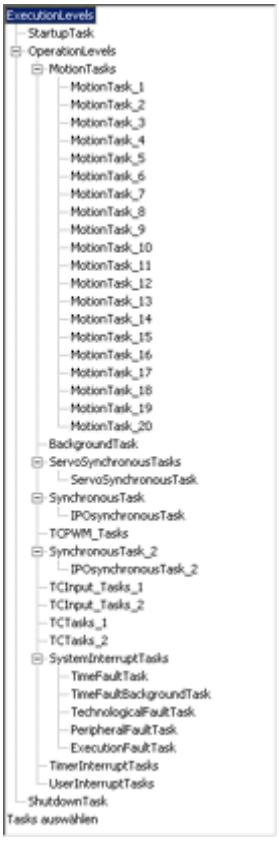


Figure 7-3 Overview of the tasks in SCOUT

Execution system

The execution system with the execution levels and the assigned tasks are displayed here.

Execution levels tree

The execution levels tree displays the available execution levels / tasks as fixed entries.

The **OperationLevels** folder contains the tasks that are available in the **RUN** mode.

- To open the **OperationLevels** folder, click the plus sign in front of it.

The list below each execution level or task name shows the configured tasks and the programs assigned to them.

After you have assigned programs to tasks, they are displayed in the execution levels tree.

Use task in execution system

Select the tasks from the list that should be used in the execution system.

- Activate the checkbox of the respective task.

Only those tasks that have been selected will be shown in the execution levels tree.

Execution system - context menu

You can select the following functions:

Function		Meaning/Note
Open		Use this to open the execution level configuration.
Expert		
	Set system cycle clocks	Use this to set the ratio between the system cycle clocks (e.g. between the interpolator cycle clock and the position control cycle clock) depending on a parameterized isochronous operation at the PROFIBUS or PROFINET interface.
Print		Use this to print the contents of the execution system. All tasks with the associated configuration are printed.
Print preview		Use this to open the print preview for the execution system.

7.1.3 Task priorities

Tasks and the programs assigned to them are started and executed at a certain time. If several tasks are to be started and executed at the same time, the task priority determines which task will be executed first.

Table 7-2 Overview of the task priorities

Priority	Execution level	Task	Cyclic/ sequential	Meaning
High	Servo, DP communication	-	z	
	Servo/T1 (DCC)	**Servodcc** **ServoSynchronousTask** **ServoTask**	z	Task of a DCC runtime group (T1) User program task in the servo cycle clock System task
	IPO/T2 (DCC)	**Ipodcc**	z	Task of a DCC runtime group (T2)
		IPOSynchronousTask • Check UserInterrupt • User program	z	User program task in the interpolator cycle clock
		IPOTask • Check the condition for WAITFORCONDITION (IPO, system)	z	System task in the IPO cycle clock
	IPO_2/T3 (DCC)	**Ipodcc_2**	z	Task of a DCC runtime group (T3)
		IPOSynchronousTask_2	z	User program task in the interpolator cycle clock 2
		IPOTask2	z	System task in the IPO cycle clock 2
	TControlTasks	**PWMsynchronousTask**	z	Temperature controller task
		PWMTask	z	System task
		InputTask_1	z	System task
		InputSynchronousTask_ 1	z	Temperature controller task
		InputTask_2	z	System task
		InputSynchronousTask_ 2	z	Temperature controller task
		ControlTask_1	z	System task
		PostControlTask_1	z	Temperature controller task
		ControlTask_2	z	System task
		PostControlTask_2	z	Temperature controller task
	DccAux (DCC)	**Dccaux**	z	Task of a DCC runtime group (T4)
	DccAux_2 (DCC)	**Dccaux_2**	z	Task of a DCC runtime group (T5)
	SystemInterruptTasks	**TimeFaultTask**	s	System alarm (in the order of events)
		TimeFaultBackgroundTask	s	System alarm (in the order of events)
		TechnologicalFaultTask	s	System alarm (in the order of events)
		PeripheralFaultTask	s	System alarm (in the order of events)
		ExecutionFaultTask	s	System alarm (in the order of events)
	UserInterruptTasks	**UserInterruptTask_1**	s	User alarm (in the order of events)
		UserInterruptTask_2	s	User alarm (in the order of events)

Priority	Execution level	Task	Cyclic/ sequential	Meaning
	TimerInterruptTasks	**TimerInterruptTask1 ... TimerInterruptTask5**	z ... z	Timer alarms
	Round robin	**BackgroundTask**	z	User program task (the sequence cannot be specified)
		MotionTask_1 ... MotionTask_32	s ... s	User program tasks (the sequence cannot be specified)
Low				

Priorities

The priority of a task *cannot* be changed by the user. For information on the execution levels of Servo_fast and IPO_fast, see also Priorities and sequences of synchronous tasks (Page 270).

Note

Tasks run according to their priorities. Thus, higher priority tasks supersede lower priority tasks.

Therefore, fluctuations of the cycle time can occur for lower priority tasks.

- **TimerInterruptTasks:**
  The shorter a time slice, the higher its priority.
- **UserInterruptTasks:**
  All tasks have the same priority and are executed in the order of their activation events.
- **Wait for condition / WAITFORCONDITION** temporarily increases the priority of a MotionTask:
  - The condition is checked with the same priority as for **UserInterruptTasks**.
  - If the condition is satisfied, the MotionTask (that was previously pending) is reactivated.
  - The commands enclosed between **WAITFORCONDITION** and **ENDWAITFORCONDITION** are executed with increased priority (between **SystemInterruptTasks** and **TimerInterruptTasks**).

Further information on **Wait for condition** / **WAITFORCONDITION** can be found in the **SIMOTION MCC** or **SIMOTION ST** programming manuals.

Checking the condition for UserInterruptTask

The condition for the UserInterruptTasks is checked in the IPOSynchronousTask, prior to the execution of the user program of the IPOSynchronousTask.

Any errors that occur for the UserInterruptTask condition will be handled as if they came from the IPOsynchronousTask.

The time required for checking the condition of the UserInterruptTask is part of the runtime of the IPOsynchronousTask.

Checking the wait condition

If wait conditions can be used, such as **Wait for condition** / **WAITFORCONDITION**, the system performs the check, the task is not started for the check.

A synchronous setting on the command (e.g. Wait until motion end) is reasonable especially in sequential tasks.

The WAITFORCONDITION condition is checked after the user program for the IPOSynchronousTask at the start of the IPOTask.

Any errors that occur for WAITFORCONDITION will be handled as if they came from the associated MotionTask.

The time required for checking the WAITFORCONDITION condition(s) is part of the runtime of the IPOTask.

Task start sequence

When the StartupTask is completed, **RUN** mode is reached.

The following tasks are then started:

- SynchronousTasks
- TimerInterruptTasks
- BackgroundTask
- MotionTasks for which the attribute for automatic start is set

Note

The priorities of the execution levels or their tasks do not indicate the order in which the MotionTasks, BackgroundTask and time-triggered tasks are started after **RUN** mode is reached.

Priorities of the temperature controller tasks

The PWM tasks have a separate layer; their priority is located between servo level and IPO level. In terms of priority, all other temperature controller tasks are sorted between the DccAux task and the Fault tasks and are located on the same layer.

7.1.4 Runtime model in SIMOTION

The following diagram shows the general sequence and (from top to bottom) the priorities of the tasks in SIMOTION.

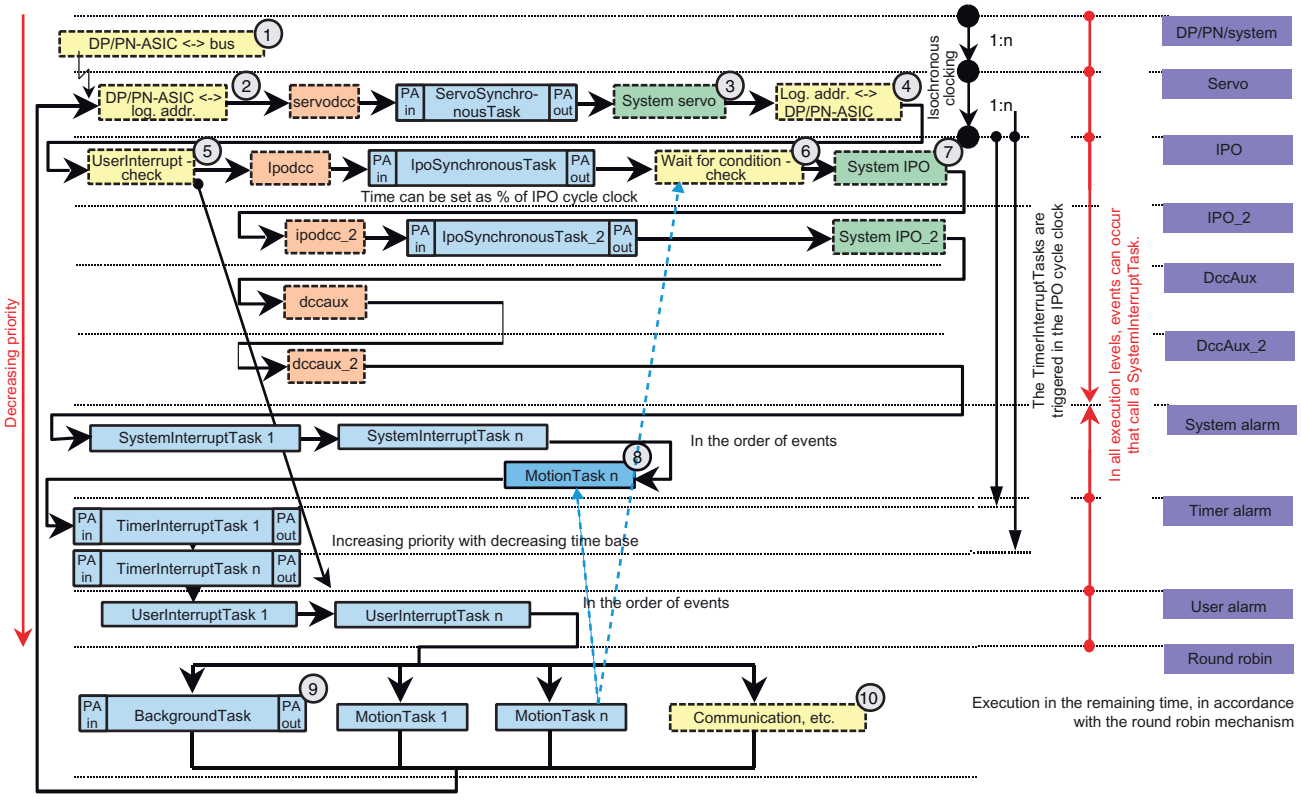


Figure 7-4 Runtime model in SIMOTION - general sequence, priorities

Explanations for the figure:

The display applies to a ratio between DP, Servo and IPO of 1:1:1

- Color blue/green: available to the user (application / technology objects)
- Color yellow (dashed): not available to the user (system tasks)

Explanation of the system tasks

1. **DP/PN-ASIC <-> Bus:**
   Data is copied from the communications chip to the PROFIBUS or PROFINET IO with IRT or fetched from there. The transfer of the I/O inputs on the logical addresses (2) is started at the end of the copy action.
   The copy action runs on its own processor and thus asynchronously to the remaining execution system. The synchronization point is the end of the data exchange between the bus and the ASIC.

2. **DP/PN-ASIC -> log. addr.:**
   The I/O inputs are loaded from the communications chip.

3. **System servo:**
   System calculations in the servo cycle clock (position controller, etc.).
   If all tasks of the servo level cannot be calculated in a single cycle (bus cycle clock), a level overflow occurs and the system enters STOP mode, the start-up lock is set and a corresponding entry is made in the diagnostic buffer. Only after a ramp-up (power on/off) or download can the system return to RUN mode.
4. **Log. addr. -> DP/PN-ASIC:**
   The I/O outputs are written to the communications chip.
5. **UserInterrupt - Check:**
   The conditions of the two user interrupts are checked.
6. **Wait for condition - Check:**
   The conditions for WAITFORCONDITION (wait for axis, wait for signal, etc.) are checked.
7. **System IPO/IPO_2:**
   The system-side components of the IPO cycle clock are calculated (motion control: Positioning profiles, synchronous operation, etc.).
8. **MotionTask n:**
   A MotionTask that is waiting with a WAITFORCONDITION will be switched in preference when the condition occurs (with higher priority).
9. **BackgroundTask:**
   The **BackgroundTask** runs here.
   The updating of the background process image (PI) is performed at the start and after completion of the complete BackgroundTask.
   The runtime model normally runs several times between the reading of the input image and the writing of the output image. I.e. depending on the size of the user program, the BackgroundTask is interrupted several times by higher priority tasks (starting with the ServoTask).
10. **Communication:**
    Communication functions (HMI, PG/PC, etc.).

See also

SynchronousTasks (Page 224)

BackgroundTask (Page 218)

7.1.5 Execution system in the symbol browser

Description

You can monitor and, if required, change program instance data of programs that are used in the execution system. This data includes the values of local variables that are normally not monitored on the unit.

For example, you have used a function block several times in the execution system on MotionTasks. You can then view the program instance data under every MotionTask in the symbol browser of the execution system.

To display the symbol browser for the execution system:

1. Select the execution system in the project navigator.
2. Click the symbol browser into the foreground in the detail view.
   The available program instance data is then displayed for each task.

Note

If you have selected "Only create program instance data once" as the compiler option, the program instance data is no longer displayed. It is displayed locally on the unit.

7.1.6 Synchronism of all components

Description

All components (the SIMOTION system, user programs, drives, bus system, and accurately-timed I/O modules) work deterministically and in strict synchronism. The individual components are synchronized via an isochronous fieldbus (PROFIBUS or PROFINET).

SIMOTION synchronizes itself with the bus cycle clock and passes this on to a second bus as well.

This ensures determinism for all the components, even if multiple machine modules and bus segments are being used. Distributed synchronous operation is also possible, e.g. as shown in the figure below:

Although axes 1 to 4 and axes 5 and 6 are controlled by different SIMOTION devices, they can still be operated synchronously.

This consistent synchronism results in high dynamic response and precision throughout the entire machine.

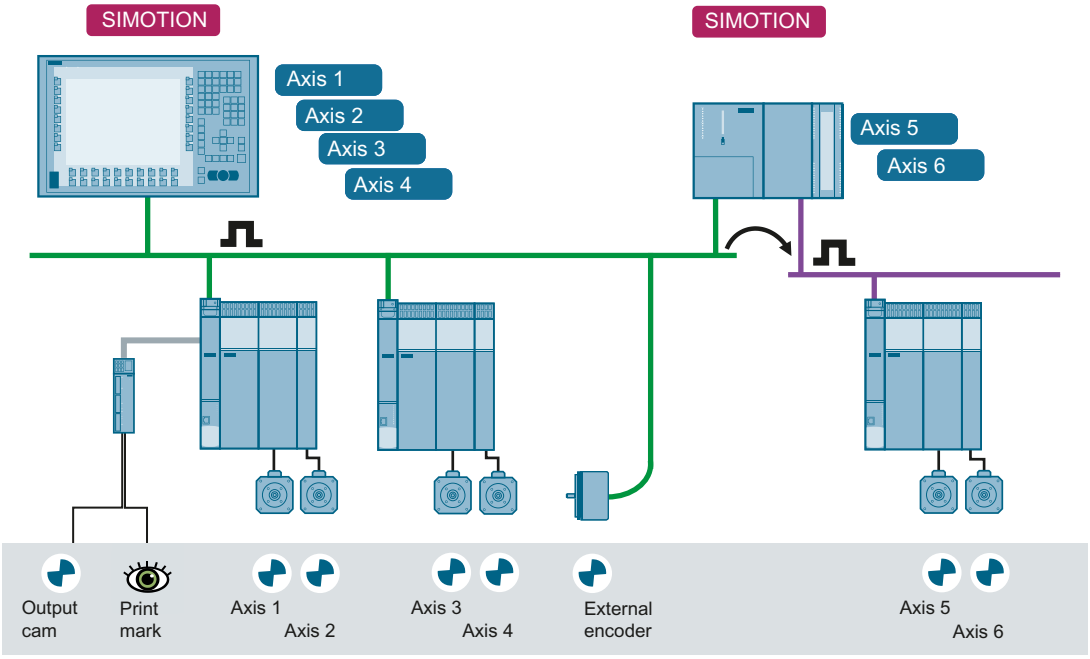


Figure 7-5 Synchronism with SIMOTION

7.1.7 Identifying the task name and type (as of Kernel V4.5)

With SIMOTION Kernel version V4.5 and higher, you can identify during runtime the name and type of the task assigned to a program.

To identify the task name, use the _getCurrentTaskName() function which is called without specification of a parameter. The return value of data type STRING contains the task name. A description of the function can be found in the "System Functions/Variables Devices" List Manual (reference list).

The task type is specified in the TaskStartInfo TSI#taskType. This TaskStartInfo contains the type of the task as an EnumTaskType. This data type is described in the TaskStartInfo of the StartupTask (Page 161).

7.2 Description of the user program tasks

7.2.1 StartupTask

The **StartupTask** is provided for the one-time initialization and the resetting of the technology objects.

It is active when the operating mode switches from **STOP** or **STOP U** to **RUN**.

It must *not* be used for process start-up or homing or for setting up axes (motion commands must not be used).

While the StartupTask is being executed, no other user program tasks except for the **SystemInterruptTask** and the **UserInterruptTask** are active.

Access to process image and symbolic I/O variables is restricted. The process image of the inputs is updated before the startup task and remains constant for the duration of the startup task. The process image of the outputs is set to zero before the startup task, and output after the startup task. Direct accesses to inputs provide the current values. Write I/O variables can be set to initial values. These values, however, act only after the startup task with the enable of all outputs on the terminals.

When the StartupTask is completed, **RUN** mode has been reached. The following tasks are now started:

- **SynchronousTasks**
- **TimerInterruptTasks**
- **BackgroundTask**
- **MotionTasks**, in which the automatic start attribute is set

Note

The priorities of the execution levels or their tasks do not indicate the order in which the MotionTasks, BackgroundTask and time-triggered tasks are started after **RUN** mode is reached.

Select the **Program assignment** tab to assign the created and compiled programs to the StartupTask and define their execution sequence.

Configuring the StartupTask

1. Click **StartupTask** in the Execution Levels tree.

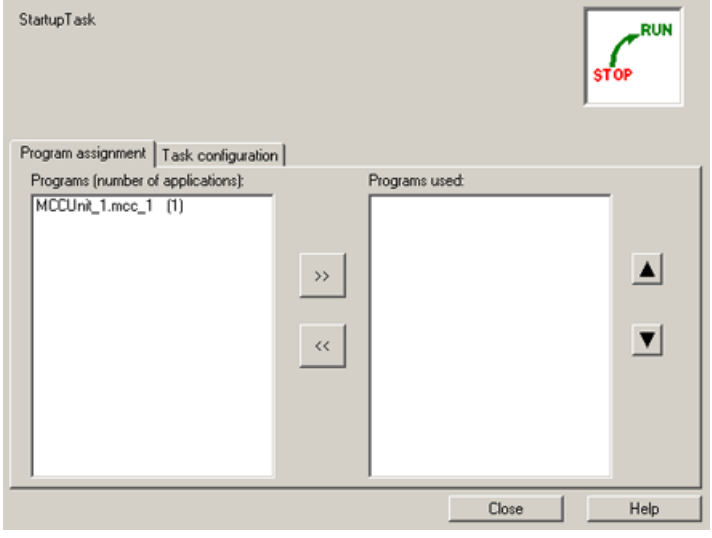


Figure 7-6 Configuration of the StartupTask (program assignment)

2. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.

3. Switch to the **Task configuration** tab.

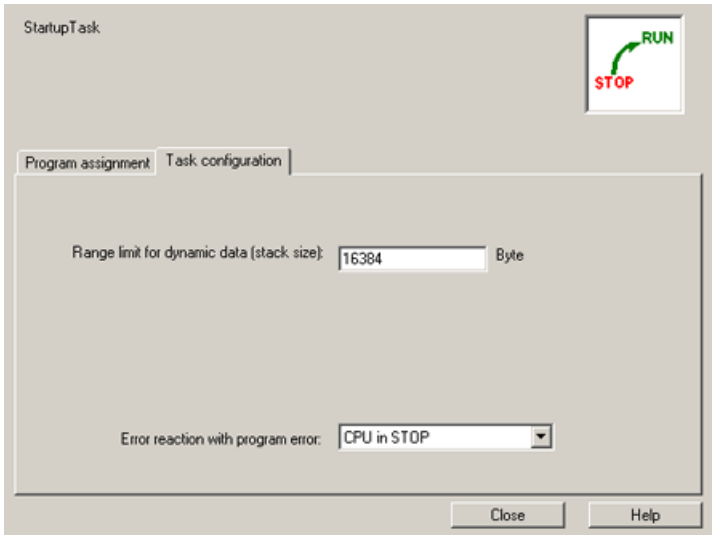


Figure 7-7 Configuration of the StartupTask (task configuration)

4. If required, enter the **Range limit for dynamic data** (**stack size**).
5. Specify the **Error response to program error** (e.g. **ExecutionFaultTask**).

Task configuration - StartupTask

In the **Task configuration** tab, parameterize the error response to program errors.

You can set the following parameters:

Field/button		Meaning/information
Range limit for dynamic data (stack size)		Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. If no programs are assigned, the CPU switches to **STOP** mode. The task in which the error occurred is terminated.

See also

Assigning programs to the execution levels/tasks (Page 244)

SystemInterruptTasks (Page 232)

7.2.2 MotionTasks

MotionTasks are intended for the programming of sequences, for programmed motion control or other sequential executions.

Example for the sequential execution: An axis traverses to a target position, waits for an enable signal, and then traverses to the next target position.

Several MotionTasks are available:

- With SIMOTION D and SIMOTION P only: 32 MotionTasks **MotionTask_1** to **MotionTask_32**

The names of the MotionTasks can be changed, see below.

MotionTasks are executed in the round robin execution level.

Note

C2xx devices only have 20 MotionTasks; all other CPUs (see above) have 32. With the C2xx, the excess MotionTasks may be deleted. Once deleted, these MotionTasks cannot be recreated.

MotionTasks and **BackgroundTask** share the free time apart from the higher-priority system and user program tasks. The relationship of the time slices between both levels can be parameterized, see Setting the time allocation (Page 276).

There is no fixed sequence of execution for MotionTasks and BackgroundTask.

Information on the influencing of the task execution is provided in Overview of the task control commands (Page 338).

Starting a MotionTask

MotionTasks are usually controlled from the user program via task control commands such as **_startTaskID**, **_stopTaskID,**, ... With the corresponding configuration (set attribute), a MotionTask starts automatically when the **RUN** mode has been reached. You can scan the current task status using the **_getStateOfTaskID** system command.

A MotionTask does not have a time monitoring, i.e. once a MotionTask is started, it can remain active for an indefinite period.

A MotionTask that waits for a synchronous command remains active with regard to its status.

Completing a MotionTask

A MotionTask is completed when the task has been completed or at the transition to the **STOP** or **STOP U** mode (start of the **ShutdownTask**).

One way to automatically suspend the task is to use wait commands **Wait for condition** / **WAITFORCONDITION**.

The task is suspended when the wait command is issued. The condition specified in the command is checked in the IPO cycle clock. When the condition is fulfilled, the MotionTask is automatically resumed. Depending on where you position the wait command, you can influence the task priority when execution continues.

The commands (with MCC in the gray area behind the command) enclosed between **WAITFORCONDITION** and **ENDWAITFORCONDITION** are executed with increased priority (between **SystemInterruptTasks** and **TimerInterruptTasks**).

Select the **Program assignment** tab to assign the created and compiled programs to the MotionTasks and define the execution sequence.

You can set the following parameters:

Field/button	Meaning/information
MotionTask	Under MotionTask, select the MotionTasks you want to assign the programs to. You can assign several programs to one MotionTask.
MotionTask_1 to MotionTask_n	Predefined names of the possible MotionTasks.
Use task in execution system	Activate the checkbox to display and use the task in the execution system. If the checkbox is deactivated, you cannot assign any programs to this task.

Configuring MotionTasks

You can define which tasks are to be started automatically when the **RUN** mode is reached. Otherwise MotionTasks must be explicitly started via programmed task control commands.

1. Click **MotionTasks** in the execution level tree.
2. Select the required task in the **MotionTask** list. The task names can be changed.
3. To do so click in the **MotionTask** field and enter a new name. Umlauts and special characters are not permitted. The name is updated in the task list.

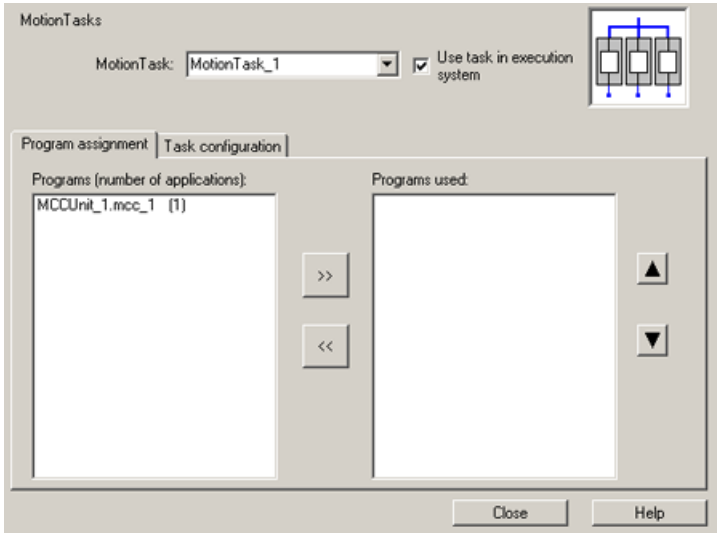


Figure 7-8 Configuration of the MotionTasks (program assignment)

4. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.

5. Select the **Task configuration** tab.

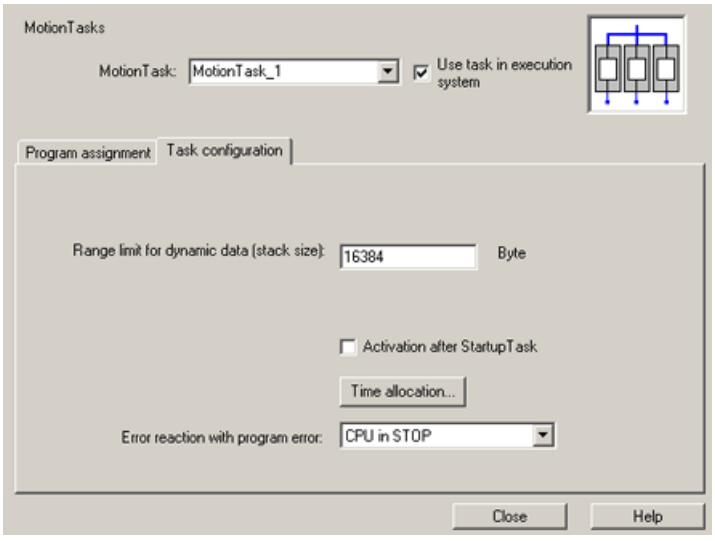


Figure 7-9 Configuration of the MotionTasks (task configuration)

6. Activate the **Activation after StartupTask** option to start the MotionTask once after the StartupTask.
7. If required, enter the **Range limit for dynamic data** (**stack size**).
8. Specify the **Error response to program error** (e.g. **ExecutionFaultTask**).
9. Repeat steps 2 through 7 for all the MotionTasks to be configured.
10. Specify the **time allocation in the round robin execution level** between MotionTasks and BackgroundTasks, see Setting the time allocation (Page 276).

Task configuration - MotionTasks

In the **Task configuration** tab, parameterize the time allocation between the MotionTasks and BackgroundTasks in the round robin execution level, and define when the MotionTasks are activated.

You can set the following parameters:

Field/button	Meaning/information
Range limit for dynamic data (stack size)	Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Activation after StartupTask	Select this if you want the MotionTask to start once when **RUN** mode is reached (StartupTask is completed).

Field/button		Meaning/information
Time allocation		Clicking this button opens the screen form for **time allocation in the round robin execution level**. This is where you can parameterize the time allocated to MotionTasks and BackgroundTasks in the round robin execution level.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. If no programs are assigned, the CPU switches to **STOP** mode. The task in which the error occurred is terminated.

See also

BackgroundTask (Page 218)

Assigning programs to the execution levels/tasks (Page 244)

SystemInterruptTasks (Page 232)

Time allocation in the round robin execution level (Page 273)

Assigning programs to the tasks (Page 320)

Specifications for the configuring (Page 320)

Description of the user program tasks (Page 212)

7.2.3 BackgroundTask

The **BackgroundTask** is provided for the programming of cyclic sequences without a fixed time frame.

It is executed cyclically in the round robin execution level, which means it will be automatically restarted on completion.

The BackgroundTask is used in programs that have to be executed cyclically, e.g. interlocking tasks, PLC tasks.

The cycle time of the BackgroundTask is monitored. When the cycle time monitoring responds, the **TimeFaultBackgroundTask** is started. The CPU will switch to **STOP** mode if the task is not configured or there is no program assigned to it.

The process image of the inputs and outputs is generated for the BackgroundTask in address space 0.0 to 63.7. The process image remains consistent during the time the BackgroundTask is being processed.

You can use the BackgroundTask to implement lower-priority, cyclical logic functions, interlocks, calculations, monitoring functions.

The **BackgroundTask** shares available CPU time with the **MotionTasks**.

Note

The BackgroundTask shares CPU time with MotionTasks and SystemTasks (e.g. communication tasks). Relative to time allocation you must consider that the runtime, i.e. the performance, is influenced by the settings (time allocation of the round robin execution level).

Starting the BackgroundTask

The BackgroundTask is started automatically when the **RUN** mode has been reached or when the task has been completed (with the next position control cycle clock).

Completion of the BackgroundTask

A BackgroundTask is completed at the transition to the **STOP** or **STOP U** mode (start of the **ShutdownTask**).

Select the **Program assignment** tab to assign the created and compiled programs to the BackgroundTasks and define the execution sequence.

Configuring the BackgroundTask

1. Click **BackgroundTask** in the Execution Levels tree.

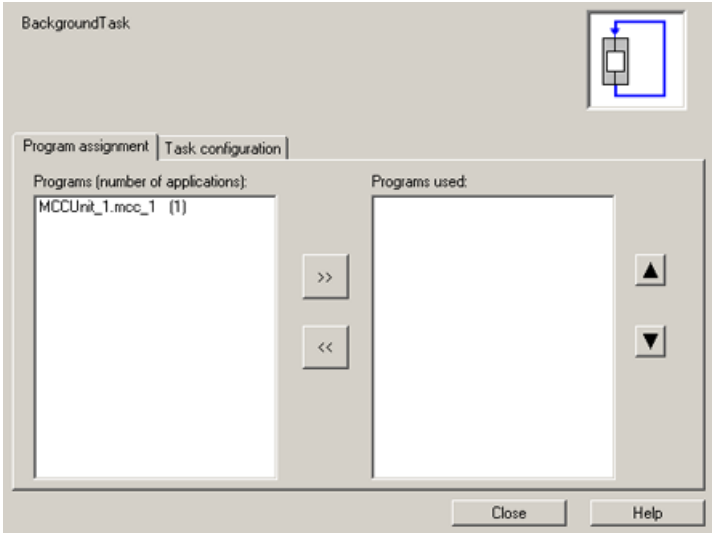


Figure 7-10 Configuration of the BackgroundTask (program assignment)

2. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.

3. Switch to the **Task configuration** tab.

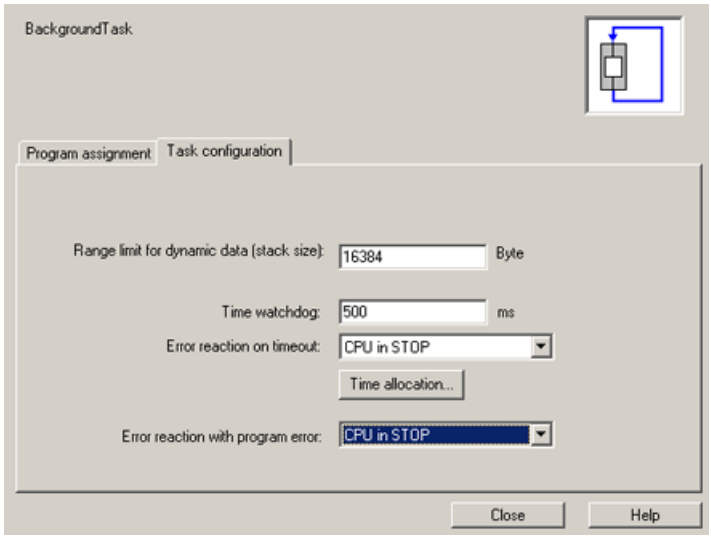


Figure 7-11 Configuration of the BackgroundTask (task assignment)

4. If required, enter the **Range limit for dynamic data (stack size)**.
5. Enter a value for the **time monitoring**.
   If this time is exceeded, the relevant SystemInterruptTask (**TimeFaultBackgroundTask**) can be called or the **CPU to STOP** can be set, 0 ms = no monitoring.
6. Specify the **Error response after timeout**.
   CPU in STOP or call **TimeFaultBackgroundTask**.
7. Specify the **time allocation in the round robin execution level** between MotionTasks and BackgroundTasks, see **Setting the time allocation**.
8. Specify the **Error response to program error** (e.g. **ExecutionFaultTask**).

Task configuration - BackgroundTask

You can parameterize the time monitoring and the error response in the **Task configuration** tab.

You can set the following parameters:

Field/button	Meaning/information
Range limit for dynamic data (stack size)	Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Time monitoring	Specify the time monitoring in ms within which the BackgroundTask must be calculated here. This allows you to specify the required cycle time for the BackgroundTask. The time monitoring is inactive if you enter 0 or no value.

Field/button		Meaning/information
Error response after time-out		You can select the error response, if the cycle for the BackgroundTask is not terminated within the time frame specified under Time monitoring.
	CPU to Stop	The CPU switches to **STOP** mode.
	TimeFaultBackground-Task	The TimeFaultBackgroundTask is started. All programs assigned to this task are started.
Time allocation		Clicking this button opens the screen form for **time allocation in the round robin execution level**. This is where you can parameterize the time allocated to MotionTasks and BackgroundTasks in the round robin execution level.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to Stop	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. The CPU subsequently changes to **STOP**.

See also

MotionTasks (Page 215)

Assigning programs to the execution levels/tasks (Page 244)

Watchdog (Page 263)

Setting of the time allocation (Page 276)

SystemInterruptTasks (Page 232)

Time allocation in the round robin execution level (Page 273)

7.2.4 TimerInterruptTasks

TimerInterruptTasks are intended for the periodical starting of programs.

Five TimerInterruptTasks, i.e. **TimerInterruptTask_1** to **TimerInterruptTask_5**, are available for different time levels.

TimerInterruptTasks are periodically started and executed in the configured fixed time frame (e.g. 100 ms).

This time frame must be a multiple of the interpolator cycle clock.

In this task, you can implement closed-loop control or monitoring functions that require a reproducible time reference without a direct link to I/O or motion control of the axes.

Additional system and user program tasks are provided for the TControl technology package. The TControl technology package is processed in the system tasks, whereas the application-specific adaptations are processed in the user program tasks.

You can find additional information in the functional description of the temperature controller, refer to the **Motion Control Additional Technology Objects** Function Manual.

Starting a TimerInterruptTask

TimerInterruptTasks are started periodically. A start delay can be set.

Completing a TimerInterruptTask

TimerInterruptTasks are completed automatically after completion of the programs assigned to the TimerInterruptTask.

You can assign the created and compiled programs to the selected TimerInterruptTask and define their execution sequence in the **Program assignment** tab.

You can set the following parameters:

Field/button	Meaning/information
For task	Use this to select one of the five TimerInterruptTasks to which you want to assign the programs. You can assign several programs to one TimerInterruptTask.
TimerInterruptTask_1 to **TimerInterruptTask_5**	Predefined names of the five TimerInterruptTasks.
Defined time level	Use this to select the time frame for restarting the TimerInterrupt-Task. You can choose time levels from the available list or enter additional ones as required. The value must be a multiple of the IPO cycle clock.
Use task in execution system	Activate the checkbox to display and use the task in the execution system. If the checkbox is deactivated, you cannot assign any programs to this task.

Configuring TimerInterruptTasks

1. Click **TimerInterruptTasks** in the Execution Levels tree.
2. Select the required task in the **For task** selection box. The names (**TimerInterruptTask_1** to **TimerInterruptTask_5**) are preset.
3. Activate the **Use task in execution system** checkbox.

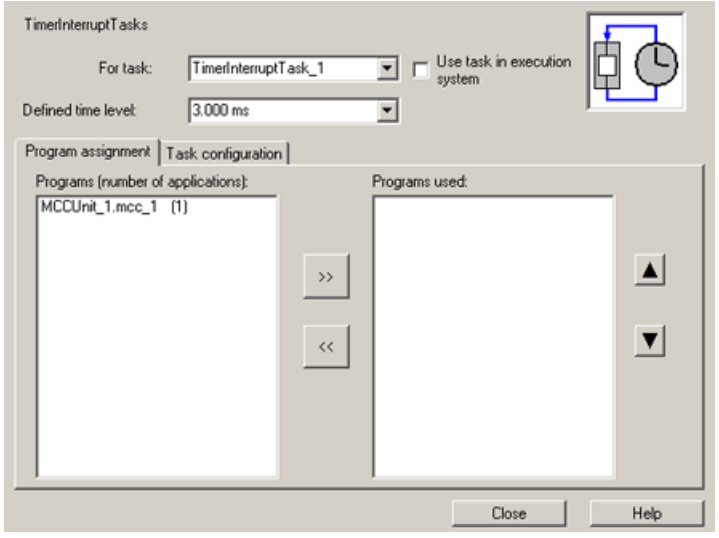


Figure 7-12 Configuration of the TimerInterruptTasks (program assignment)

4. Select a **Defined time level** or enter an arbitrary integer value. This value must be an integral multiple of the interpolator cycle clocks (IPO cycle clocks). Different settings are rounded up to the next integral multiple of the IPO cycle clock.
5. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.
6. Switch to the **Task configuration** tab.

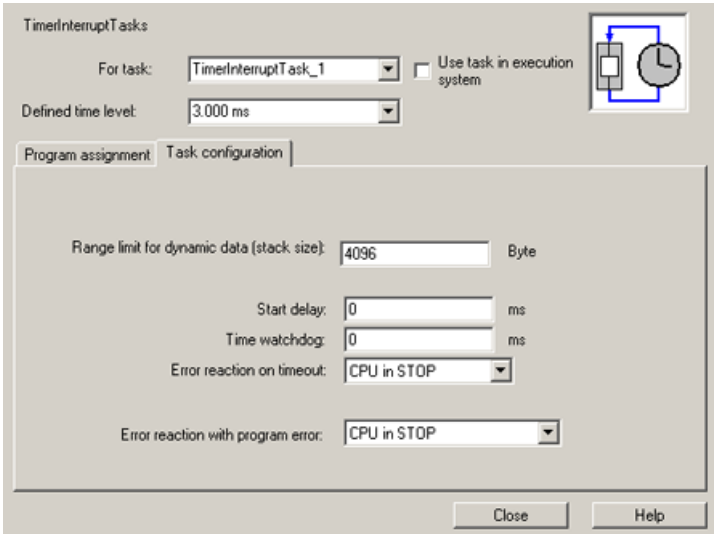


Figure 7-13 Configuration of the TimerInterruptTasks (task configuration)

7. If required, enter the **Range limit for dynamic data (stack size)**.
8. Enter the **Start delay** for this task, if applicable.
   To prevent different time-triggered tasks from starting at the same time, you can define a start delay for each task. This prevents different TimerInterruptTasks from being executed at the same time.
   Example:
   IPO cycle clock: 4 ms
   TimerInterruptTask_1: 8 ms
   TimerInterruptTask_2: 16 ms
   Both TimerInterruptTasks are started simultaneously in every fourth IPO cycle clock. By delaying the start of one of the two tasks by 4 ms, you can prevent them from being executed at the same time. In this way, the load on the system is distributed more evenly, and the behavior of the lower-priority tasks can be reproduced.
9. Enter a value for the **time monitoring**.
   If this time is exceeded, the associated SystemInterruptTask (**TimeFaultTask**) can be called or the **CPU to STOP** can be set, 0 ms = no monitoring.
10. Specify the **Error response with timeout**.
    CPU in STOP or call **TimeFaultTask**.
11. Specify the **Error response to program error** (e.g. **ExecutionFaultTask**).
12. Repeat steps 3 to 10 for all the TimerInterruptTasks to be configured.

Task configuration - TimerInterruptTasks

You can define the start delay, time monitoring and the error response for the TimerInterruptTasks in the **Task configuration** tab.

You can set the following parameters:

Field/button		Meaning/information
Range limit for dynamic data (stack size)		Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Start delay		Enter a time value in ms for the task start delay. The start of TimerInterruptTask is delayed by this time value. This delay ensures that different TimerInterruptTasks will not start simultaneously, resulting in a timeout.
Time monitoring		Specify the cycle time in ms for executing the TimerInterruptTask. Enter a value for time monitoring. The time monitoring is inactive if you enter 0 or no value.
Error response after timeout		You can select the error response if the TimerInterruptTask is not terminated within the time frame specified under Time monitoring.
	CPU to Stop	The CPU switches to **STOP** mode.
	TimeFaultTask	The TimeFaultTask is started. All programs assigned to this task are started.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. If no programs are assigned, the CPU switches to **STOP** mode. The task in which the error occurred is terminated.

See also

Assigning programs to the execution levels/tasks (Page 244)

Watchdog (Page 263)

SystemInterruptTasks (Page 232)

7.2.5 SynchronousTasks

SynchronousTasks are started synchronously to the specified system cycle clock on a periodic basis. They run with high priority in the time-triggered execution level.

The following SynchronousTasks are available:

- **ServoSynchronousTasks**
  - **ServoSynchronousTask** (as of V4.0): Synchronous with the position control cycle clock.
  - **ServoSynchronousTask_fast** (as of V4.2): Synchronous with the Servo_fast_clock_cycle (optional, only available for specific SIMOTION modules, see Servo_fast (application cycle clock for fast bus system) (Page 264)).

  In the servo-synchronous tasks, you can implement time-critical terminal - terminal responses for I/O or fast influencing of setpoints on the servo level.
  See also isochronous I/O processing on fieldbus systems (Page 283).
  Application, such as for fast terminal - terminal response. If a TO is configured for execution in the position control cycle clock then TO and motion commands can also be issued.
- **IPOsynchronousTasks**
  - **IPOsynchronousTask**: Synchronous with interpolator cycle clock IPO.
  - **IPOsynchronousTask_2**: Synchronous with interpolator cycle clock IPO_2
  - **IPOSynchronousTask_fast** (as of V4.2): Synchronous with the interpolator cycle clock IPO_fast (optional, only available for specific SIMOTION modules, see Servo_fast (application cycle clock for fast bus system) (Page 264)).

  In IPO-synchronous tasks, you can implement time-critical functions that have a direct effect on the functions of the technology object. The user program is executed prior to the interpolator, i.e. the programmed functions can be effective in the same IPO cycle clock.
  Application, e.g. for a fast start depending on the event, fast response to events terminal - axis

Additionally, there are further SynchronousTasks for TControl:

- **PWMsynchronousTask**: Synchronous with the PWM cycle clock
- **InputSynchronousTask_1/InputSynchronousTask_2**: Synchronous with the Input1/2 cycle clock before temperature control
- **PostControlTask_1/PostControlTask_2**: Synchronous with the Control1/2 cycle clock after temperature control

You can assign the created and compiled programs to the selected SynchronousTask and define their execution sequence on the **Program assignment** tab.

You can set the following parameters:

Field/button		Meaning/information
For cycle clock level		Use this to select the SynchronousTask to which you want to assign the programs. You can assign several programs to one SynchronousTask.
	ServoSynchronousTask_fast	Optional. SynchronousTask which is started in the application cycle for the fast bus system (Servo_fast).
	IPOsynchronousTask_fast	Optional. SynchronousTask which is started in the interpolator cycle for the fast bus system (IPO_fast).
	ServoSynchronousTask	SynchronousTask which is started in the position control cycle clock
	IPOsynchronousTask	SynchronousTask which is started in the interpolator cycle clock (IPO cycle clock).
	IPOsynchronousTask_2	SynchronousTask which is started in the second interpolator cycle clock (IPO2 cycle clock).
	PWMsynchronousTask (TCPWM_Tasks)	SynchronousTask which is used for the actuating signal output of the TO temperature controller (temperature channel). Defines the basic cycle clock for further tasks of the TO temperature controller.
	InputSynchronousTask_ 1/2 (TCInput_Tasks_1/2)	SynchronousTask which is used for the actual value measurement of the TO temperature controller.
	PostControlTask _1/2 (TCTasks_1/2)	SynchronousTask which is used for the control of the TO temperature controller.
Use task in execution system		Activate the checkbox to display and use the task in the execution system. If the checkbox is deactivated, you cannot assign any programs to this task.

ServoSynchronousTask/ServoSynchronousTask_fast

ServoSynchronousTasks are intended for applications in which fast processes are to be implemented using isochronous mode.

ServoSynchronousTasks run within a position control cycle clock and behave similar to IPOsynchronousTasks.

ServoSynchronousTasks_fast run within a Servo_fast cycle clock and behave similar to the ServoSynchronousTasks.

The ServoSynchronousTasks are suitable for:

- Fast responses (e.g. for short terminal-terminal times for I/O processing)
- Influencing servo-effective system variables
  Information on servo effectiveness of the system variables is provided in the SIMOTION reference list technology package CAM system variables.
- For motion commands if a TO (in exceptional cases) is configured for execution in the position control cycle clock.

They are not suitable for:

- Communication
- For motion commands if a TO is configured for execution in the IPO cycle clock (since this is evaluated in the IPO cycle clock)

If a TO is configured for execution in the position control cycle clock then TO and motion commands can also be issued.

The same characteristics apply as for the IPOSynchronousTasks.

The ServoSynchronousTask can be configured in the execution system. The cycle time of the servo execution level is set with the servo. The default setting is active.

The ServoSynchronousTask_fast can be configured in the execution system. The default setting is inactive.

There is a process image available for the ServoSynchronousTask as for the other cyclic tasks. This only results in an improved performance if the same I/O addresses are accessed several times.

As of V4.1, the monitoring of the ServoSynchronousTasks can also be set. In this way, MOTION commands are also permitted in the ServoSynchronousTask, where the IPO share is executed in the servo (all commands with the step enabling condition IMMEDIATELY).

You can set the following parameters:

Field/button		Meaning/information
Range limit for dynamic data (stack size)		Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Monitoring when executing synchronous functions		You can select the error response if the ServoSynchronousTask is not completed within the system cycle clock.
	Discontinue time monitoring and interrupt the task	Compatible with the functionality up to V4.0 With synchronous functions, time monitoring is suspended and one servo cycle clock is lost - all old projects have this setting, too, if they have been updated to V4.1 (CPU replacement).
	Leave time monitoring active	Time monitoring is not suspended, i.e. the CPU goes to **STOP** with timeout when synchronous functions are called which really interrupt the ServoSynchronousTask. If overflows are tolerated, it might be possible to avoid **STOP** - this is the default setting for newly created CPUs. Diagnostics buffer entry: "STOP by execution system, cause: Timeout".
	CPU goes to STOP	There is no synchronous function permitted. In the case of a synchronous function, the CPU goes to **STOP** - even if the ServoSynchronousTask is not interrupted in the specific case. Diagnostics buffer entry: "Impermissible calling of a system/package function".

Field/button		Meaning/information
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. If no programs are assigned, the CPU switches to **STOP** mode. The task in which the error occurred is terminated.

IPOsynchronousTask/IPOsynchronousTask_2/IPOsynchronousTask_fast

IPOSynchronousTasks are intended for applications that require, for example, fast and deterministic responses or correction movements. Optimum time transfer of motion commands to the motion control system is thus possible.

IPOsynchronousTasks run immediately before the interpolator within an IPO cycle clock. Thus, commands in these tasks can directly affect the motion control.

The **IPOSynchronousTask** is started synchronously to the IPO_clock_cycle while the **IPOSynchronousTask_2** is started synchronously to the IPO_clock_cycle_2, i.e. a reduced IPO cycle clock.

The **IPOSynchronousTask** is executed before the internal IPOTask and **IPOsynchronousTask_2** is executed before IPOTask_2.

The **IPOSynchronousTask_fast** is started synchronously with IPO_fast and, depending on the version, before the position control cycle clock.

The following properties must be specified for the **IPOsynchronousTasks**:

- Task configuration
- Number of level overflows in the IPO/IPO_2 cycle clock
  Not for IPOsynchronousTask_fast
  If the tasks in the IPO level (**IPOSynchronousTask**, **IPOTask**) are not completed within an IPO cycle clock, an overflow will occur. An overflow of a task in the cycle clock (n) must be processed in the following cycle clock (n+1).
  The same applies for the tasks in the IPO_2 level (**IPOSynchronousTask_2**, **IPOTask_2**).
  You can set the number of overflows which should be tolerated in sequence (n = 0 ... 5). The ratio can be violated n-times. The internal monitoring counter is reset when a task has been performed without overflowing.
  The accumulated level overflows are displayed in the **numberOfSummarizedTaskOverflow** system variables.
  No level overflows are permitted for **ServosynchronousTask_fast** and **IPOSynchronousTask_fast**.

- Error response after timeout (level overflow): The CPU goes to **STOP** mode and a starting lockout is set.
- **IPOsynchronousTask/IPO-clock_cycle or IPOsynchronousTask_2/IPO_2-clock_cycle**:
  Time ratio between IPOsynchronousTask and IPO cycle clock or IPOsynchronousTask_2 and IPO_2 cycle clock.
  A timeout occurs if an IPO-synchronous task exceeds this ratio.
  You also define the time frame for the IPOTask in the system cycle clocks. In the IPOSynchronousTask/IPOClockCycle parameter, you can specify the percentage of time that is to be made available for the IPOSynchronousTask per cycle clock.
  If, for example, 4 ms is set as IPO cycle clock and 25% set as ratio, the runtime of the IPOSynchronousTask must not exceed 1 ms, including possible interrupts due to higher priority tasks.
  Recommendation: With a gear ratio of servo: IPO > 1, enter a percentage as high as possible.

Note

If a synchronous system function is called that suspends the IPOSynchronousTask, the CPU will go into **STOP** mode with a diagnostic buffer entry.

- It is possible to configure whether time monitoring is to be performed.
- Note that the copy times of the non-equidistant I/O data are included in the runtime of the IPO/IPO_2 or the IPO-synchronous user tasks.

Configuring SynchronousTasks

1. Click the appropriate task in the Execution Level tree.
2. Select the required SynchronousTask in the **For cycle clock level** selection box.
3. Activate the **Use task in execution system** checkbox.

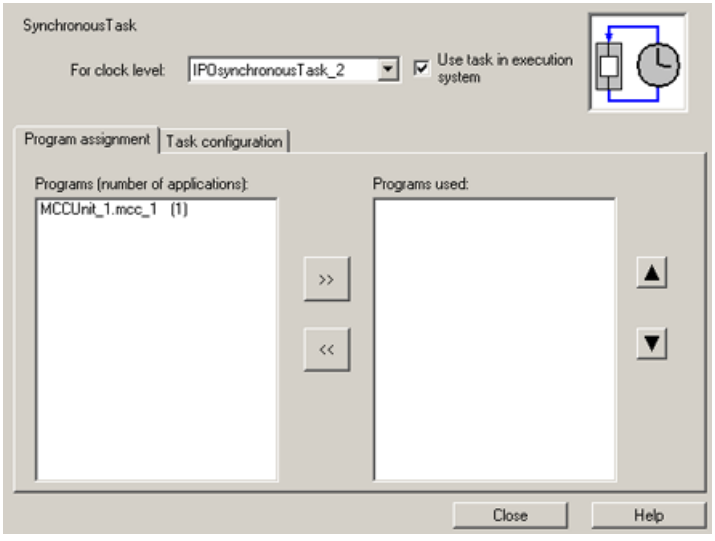


Figure 7-14 Configuration of the SynchronousTasks (program assignment)

4. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.

5. Switch to the **Task configuration** tab.

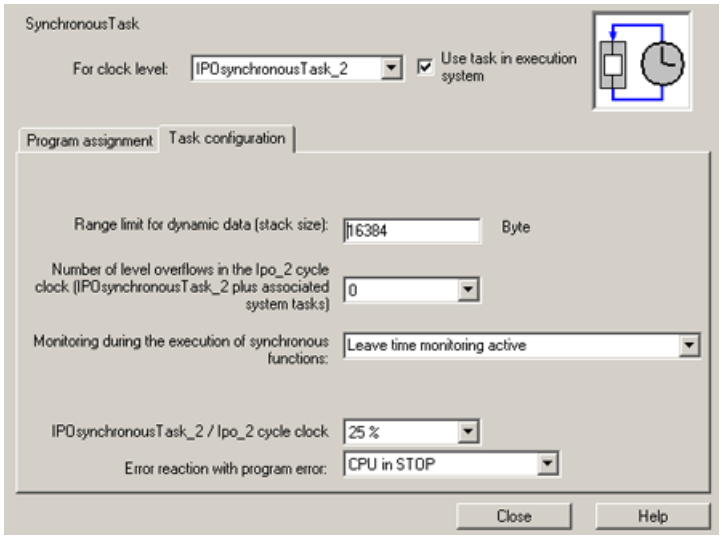


Figure 7-15 Example: Configuration of the IPOsynchronousTask (task configuration)

6. The following options are also available:
   - **Stack size** as an option in the Task configuration tab
   - Specify the number (0-5) of permitted level overflows in the **Level overflows** selection box
     No level overflows are permitted for ServoSynchronousTask_fast and IPOsynchronousTask_fast.
   - Select the ratio between SynchronousTask and IPO task
   - Select the **Error response to program error**
   - Configure the IPOSynchronousTask/IPOTask time monitoring
7. Repeat steps 3 through 5 for all the synchronous tasks to be configured.

Task configuration - SynchronousTasks

On the **Task configuration** tab you can select the error response for the SynchronousTask.

You can set the following parameters:

Table 7-3 Task configuration IPOsynchronousTasks

Field/button		Meaning/information
Range limit for dynamic data (stack size)		Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task.
Number of level overflows in the IPO cycle clock or IPO2 cycle clock		Not for IPOsynchronousTask_fast. Enter the number of tolerable level overflows in the IPO/IPO_2 cycle clock for the IPOsynchronousTasks. If the number of overflows is less than the entered overflows, there will be no error response. When arithmetic operations are processed, level overflows in the IPO/IPO_2 cycle clock can occur that can be tolerated by the user for the SynchronousTask and SynchronousTask_2 execution levels. Execution level overflows may occur if: • IPOSynchronousTask + IPO task > IPO cycle clock or • IPOSynchronousTask_2 + IPO_2 task > IPO_2 cycle clock This set tolerance level causes the next IPO cycle clock to be used for terminating the arithmetic operation of the previous cycle clock without triggering the set error response. When the set number of overflows is exceeded, or if no tolerance has been set, an entry is made in the diagnostic buffer and the error response (CPU to STOP with starting lockout) results. A maximum of 5 execution level overflows can be set.
Monitoring when executing synchronous functions		You can select the error response if the SynchronousTask is not completed within the system cycle clock.
	Discontinue time monitoring and interrupt the task	With synchronous functions, the time monitoring function is suspended and one IPO cycle is lost.
	Leave time monitoring active	Time monitoring is not suspended, i.e. the CPU goes to **STOP** with timeout when synchronous functions are called which really interrupt the IPOSynchronousTask. If IPO overflows are tolerated, it might be possible to avoid **STOP** - this is the default setting for newly created CPUs. Diagnostics buffer entry: "STOP by execution system, cause: Timeout".
	CPU goes to STOP	There is no synchronous function permitted. In the case of synchronous functions, the CPU goes to **STOP** - even if the IPOSynchronousTask is not interrupted in the concrete case. Diagnostics buffer entry: "Impermissible calling of a system/package function". The behavior applies to the IPOsynchronousTask, IPOsynchronousTask_2 and PWMTask.

Field/button		Meaning/information
IPOsynchronousTask / IPO-ClockCycle or **IPOsynchro-nousTask_2 / IPO_2Clock-Cycle**		Not for IPOsynchronousTask_fast. Here you select the duration for the specified tasks as a percentage of the cycle clock (e.g. IPOsynchronousTask / IPO cycle clock). If the task requires more time than set here, the system will respond according to the settings for error response. To configure the system cycle clocks, select the CPU in the project navi-gator and select Target system > Expert > Set system cycle clocks in the menu. **25%**: The maximum duration of the task is 25% of the cycle clock. **50%**: The maximum duration of the task is 50% of the cycle clock. **75%**: The maximum duration of the task is 75% of the cycle clock.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. If no programs are assigned, the CPU switches to **STOP** mode. The task in which the error occurred is terminated.

See also

Isochronous I/O processing on fieldbus systems (Page 283)
Timeouts and level overflows (Page 260)
Assigning programs to the execution levels/tasks (Page 244)
Sequence model for DCC blocks (DCB) (Page 294)
Setting size of local data stack (Page 597)

7.2.6 SystemInterruptTasks

SystemInterruptTasks are started and executed once when a system event occurs.

The following SystemInterruptTasks are available:

- **TimeFaultTask**: Starts in the event of a TimerInterruptTask timeout
- **TimeFaultBackgroundTask**: Starts in the event of a BackgroundTask timeout
- **TechnologicalFaultTask**: Starts in the event of an error on a technology object
- **PeripheralFaultTask**: Starts in the event of an error on the I/O
- **ExecutionFaultTask**: Starts in the event of an error when executing a program

Note

An exception in the DCC tasks does not call a SIMOTION fault task.

Starting a SystemInterruptTask

An internal system task checks approx. every 10 ms whether a configured event has occurred and then starts the applicable SystemInterruptTask automatically. The cycle clock for this internal system task is independent of the defined cycle clock for IPO or servo.

The SystemInterruptTask must be used in the execution system and a program must be assigned to this task. If this is not the case and an event occurs which starts this SystemInterruptTask, the CPU enters **STOP** state.

Up to eight different interrupts can be stored in the buffer. If another interrupt occurs, the buffer overflows and the CPU goes into **STOP** mode, too.

The events that caused a SystemInterruptTask to be called can be queried using the associated **TaskStartInfo** for this task and are described under Using Taskstartinfo (Page 160).

Completing a SystemInterruptTask

A SysteminterruptTask is completed automatically after completion of the programs assigned to the SystemInterruptTasks.

You can assign the created and compiled programs to the selected SystemInterruptTask and define their execution sequence in the **Program assignment** tab.

You can set the following parameters:

Field/button		Meaning/information
For task		Use this to select one of the SystemInterruptTasks to which you want to assign the programs. You can assign several programs to one SystemInterruptTask.
	ExecutionFaultTask	Program processing error, for example, division by zero.
	PeripheralFaultTask	I/O alarms such as process alarms, diagnostic alarms, removing and inserting modules.
	TechnologicalFaultTask	Technological alarms such as alarms, warnings and notes.
	TimeFaultBackgroundTask	Timeout in the BackgroundTask.
	TimeFaultTask	Timeouts, for example in the TimerInterruptTasks, system runtime and cycle time.
Alarm configuration		This displays the window for the configuration of the technological alarms. You can configure the alarm response to a technological alarm for the SystemInterruptTasks.
Use task in execution system		Activate the checkbox to display and use the task in the execution system. If the checkbox is deactivated, you cannot assign any programs to this task.

TimeFaultTask

The **TimeFaultTask** is started when the time monitoring of a **TimerInterruptTask** responds. The CPU will switch to **STOP** mode if the TimeFaultTask is not configured or if there is no program assigned to it.

In the **TimeFaultTask**, you can program the response to timeouts of TimerInterruptTasks.

For additional information, see Using Taskstartinfo (Page 160).

TimeFaultBackgroundTask

The **TimeFaultBackgroundTask** is started when the time monitoring of the BackgroundTask responds. The CPU will switch to **STOP** mode if the **TimeFaultBackgroundTask** is not configured or if there is no program assigned to it.

In the **TimeFaultBackgroundTask**, you can program how timeouts of **BackgroundTasks** are handled.

TechnologicalFaultTask

The **TechnologicalFaultTask** is started when the technology package generates an alarm or information message. The CPU will switch to **STOP** mode if the TechnologicalFaultTask is not configured or if there is no program assigned to it.

Generally, alarm messages have a direct effect on the behavior of the technology package and must be acknowledged before technology functions can be activated again.

In the **TechnologicalFaultTask** you can, for example, acknowledge errors directly and/or initiate further responses with respect to the sequence of operations on the machine.

For additional information, see Using Taskstartinfo (Page 160).

PeripheralFaultTask

The **PeripheralFaultTask** is started immediately in accordance with its priority for I/O access errors.

I/O access errors can occur, for example, when the load voltage supply for the I/O module fails or other errors occur on the I/O module.

For further information, refer to the descriptions of the *I/O modules*.

The task for which an error occurred during its I/O access, *will not be terminated*.

The CPU will switch to **STOP** mode if the **PeripheralFaultTask** is not configured or if there is no program assigned to it.

For additional information, see Using Taskstartinfo (Page 160).

ExecutionFaultTask

The **ExecutionFaultTask** is started immediately in accordance with its priority for program run errors.

Examples of program run errors:

- Faulty operations with floating-point numbers, such as logarithms of negative numbers, invalid numbers, etc.
- Division by zero
- Violation of array limits
- Error while accessing system variables

The task in which the error occurred is terminated.

The CPU will switch to **STOP** mode if the **ExecutionFaultTask** is not configured or if there is no program assigned to it.

The CPU to **STOP** error response is possible for all tasks and starts the **ShutdownTask**. The SIMOTION device switches to **STOP** mode.

An error response for the ExecutionFaultTask restarts the ExecutionFaultTask.

The following tasks can be restarted by commands in the program of the ExecutionFaultTask:

- StartupTask
- ShutdownTask
- MotionTasks

For the following tasks, the SIMOTION device switches to **STOP** mode once the ExecutionFaultTask is completed and the ShutdownTask is started:

- BackgroundTask
- TimerInterruptTasks
- SynchronousTasks

For additional information, see Using Taskstartinfo (Page 160).

Note

Program errors in the **ExecutionFaultTask** and in the **ShutdownTask** switch the system to **STOP** mode immediately.

Configuring SystemInterruptTasks

1. Click **SystemInterruptTasks** in the Execution Levels tree.
2. Select one of the specified tasks in the **For task** selection box.

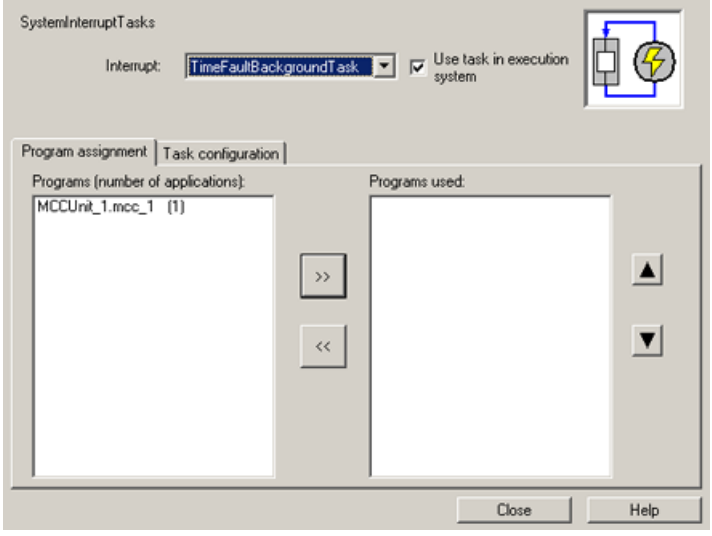


Figure 7-16 Configuration of the SystemInterruptTasks (program assignment)

3. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.

4. Switch to the **Task configuration** tab.

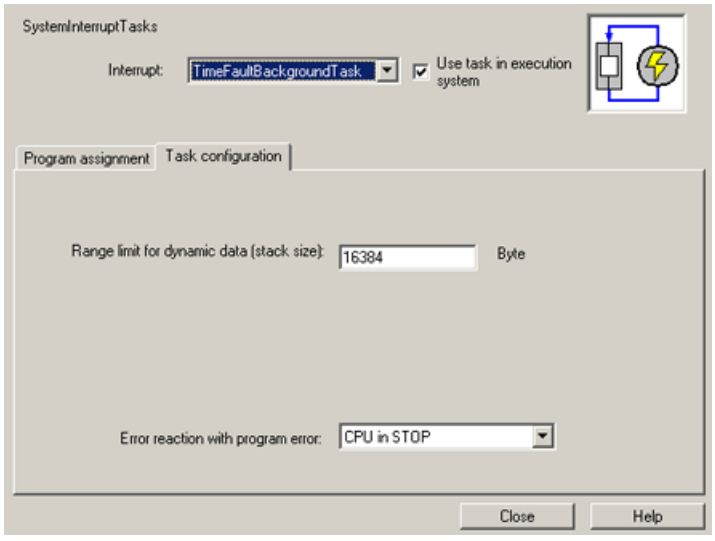


Figure 7-17 Configuration of the SystemInterruptTasks (task configuration)

5. Configure the task.
6. Repeat steps 3 to 5 for all the SystemInterruptTasks to be configured.
7. To configure the technological alarms:
   Click **Alarm configuration**.

Task configuration - SystemInterruptTasks

In the **Task configuration** tab, parameterize the error response to program errors.

You can set the following parameters:

Field/button		Meaning/information
Range limit for dynamic data (stack size)		Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. If no programs are assigned, the CPU switches to **STOP** mode. The task in which the error occurred is terminated.

See also

Information on starting a task: TaskStartInfo (TSI) (Page 263)

7.2.7 UserInterruptTasks

UserInterruptTasks are intended for user-defined actions.

Two UserInterruptTasks are available: **UserInterruptTask_1** and **UserInterruptTask_2**.

A defined condition must be specified for the UserInterruptTask. Each time the condition is fulfilled, the UserInterruptTask is started.

If you want to use a **UserInterruptTask**, the **IPOsynchronousTask** must be used in the execution system.

UserInterruptTasks are *not active* during a **StartupTask** and a **ShutDownTask**.

Starting a UserInterruptTask

UserInterruptTasks are started automatically as soon as the user-defined interrupt condition is fulfilled. The interrupt condition is checked in the interpolator cycle clock.

If both UserInterruptTasks are started at the same time, **UserInterruptTask_1** is processed before **UserInterruptTask_2**.

Note

If a user-defined interrupt occurs during shutdown, the UserInterruptTask is not started anymore.

During shutdown (ShutdownTask), starting a UserInterruptTask is only possible using the _startTaskId() command.

Completing a UserInterruptTask

UserInterruptTasks are completed automatically after completion of the programs assigned to the UserInterruptTask.

You can assign the created and compiled programs to the selected UserInterruptTask and define their execution sequence in the **Program assignment** tab.

You can set the following parameters:

Field/button		Meaning/information
For task		Use this to select one of the two UserInterruptTasks to which you want to assign the programs. You can assign several programs to one UserInterrupt-Task.
	UserInterruptTask_1 and **UserInterrupt-Task_2**	Predefined names of the UserInterruptTask.

Field/button	Meaning/information
Defined condition	Here you define the condition for the selected UserInterruptTask according to IEC 61131. During operation, the specified condition is checked in the interpolator cycle clock. If the condition is fulfilled, the UserInterruptTask is started with the assigned programs. You enter the condition in the input field by using the symbol browser (variables via drag and drop) and the **Command library** tab in the project navigator (operators via drag and drop). Only simple conditions (e.g. logic operations of inputs and local CPU variables) and Boolean variables are permissible (see below).
Use task in execution system	Activate the checkbox to display and use the task in the execution system. If the checkbox is deactivated, you cannot assign any programs to this task.

Configuring UserInterruptTasks

1. Click **UserInterruptTasks** in the Execution Levels tree.
2. Select a UserInterruptTask in the **For task** selection box.
3. Specify the condition for starting this task.

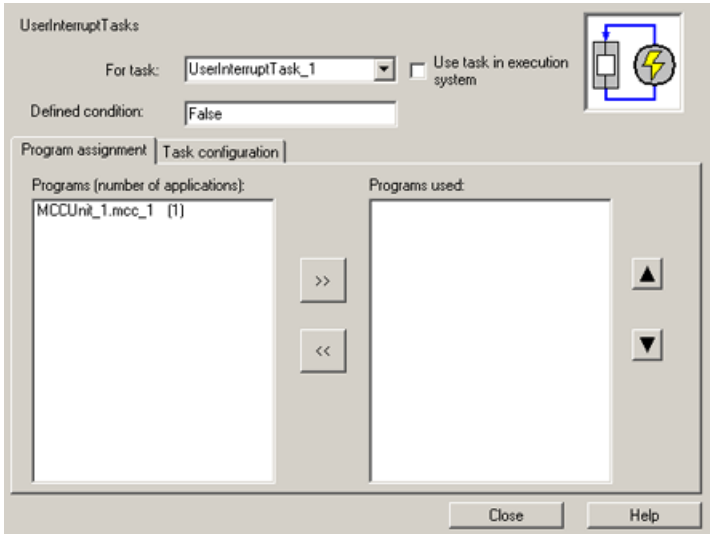


Figure 7-18 Configuration of the UserInterruptTasks (program assignment)

4. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.

5. Switch to the **Task configuration** tab.

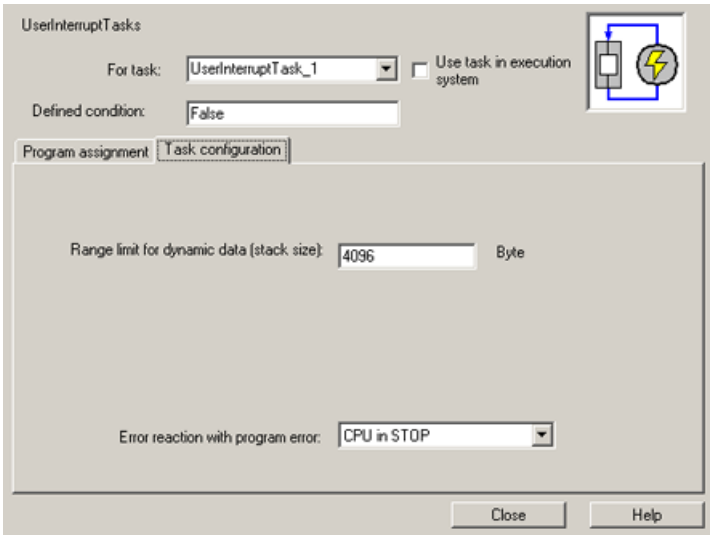


Figure 7-19 Configuration of the UserInterruptTasks (task configuration)

6. Configure the task.
7. Repeat steps 3 to 6 for the second UserInterruptTask.

Formulating a condition for a UserInterruptTask

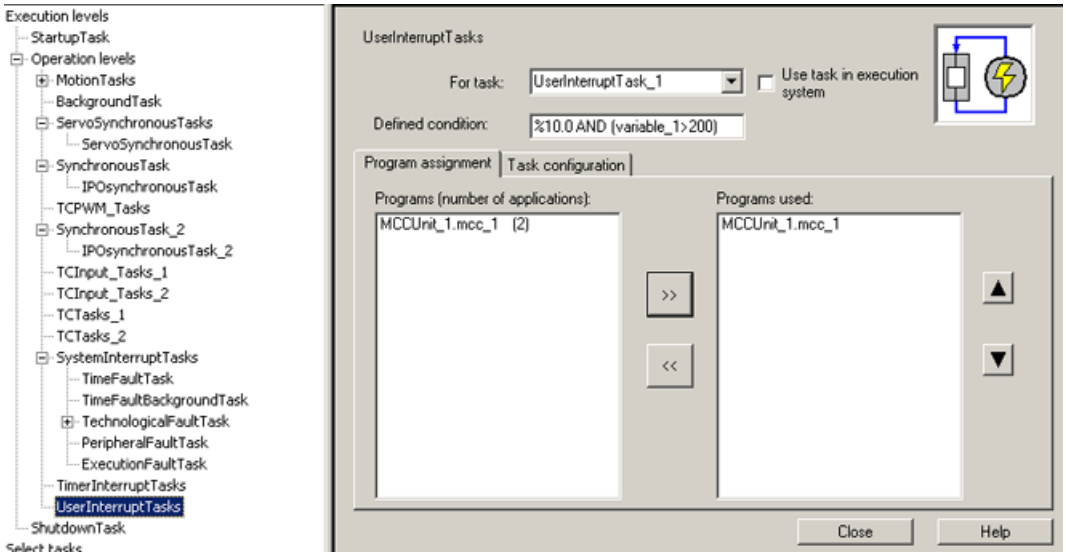


Figure legend - example:

When the digital input 0.0 = TRUE and the global variable "variable_1" has a value > 200.0, then the UserInterruptTask_1 is executed once.

Figure 7-20 Configuration of a UserInterruptTasks with condition

You enter the condition for starting a UserInterruptTask as a formula according to IEC 61131 (Structured Text). You can use the following variables:

- I/O variables
- Global device variables
- Unit variables in the interface section of a program source (ST/MCC or LAD/FBD unit). Syntax for the use of a *var_name* unit variable from the *src_name* program source:
  - Up to and including SIMOTION Kernel V4.3:
    Only specification of the variable name <VariableName>.
  - As of SIMOTION Kernel V4.4:
    The program source must also be specified in the form
    _device\<UnitName>.<VariableName>
    (example _device\mySourceFile.myVar).
    As previously, global device variables can only be specified directly via the variable names.

SIMOTION SCOUT helps you create the formula.

To formulate a condition for starting a UserInterruptTask:

1. Under **For task**, select the UserInterruptTask for which you want to define the start condition.
2. Take the required operators from the **Command library** tab and insert them in the **Defined condition** text field using drag and drop.
   Alternatively, you can enter the operators directly in the text field.
3. Take the operands from the symbol browser and insert them in the **Defined condition** text field using drag and drop or copy and paste. Alternatively, you can enter the operands directly in the text field.

Creating conditions with the command library

The **Command library** tab contains a list of mathematical operations and functions. You can use these to create conditions.

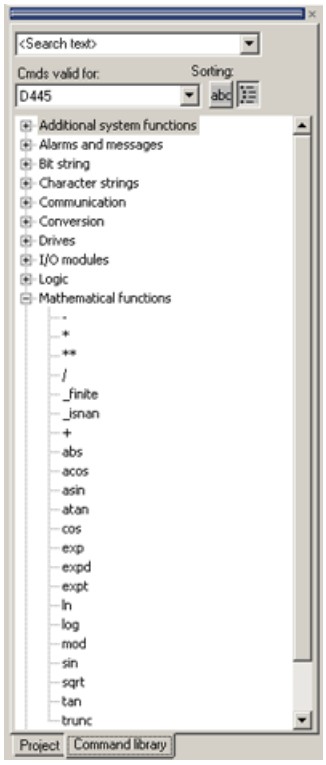


Figure 7-21 Command library in SCOUT

To open the individual operations, click the plus sign in front of the folders. You can drag the operators to the required text field via drag and drop. You can use these commands, for example, to define conditions.

Note

The following applies when updating the version of SIMOTION Kernel as of V4.4:

If you use unit variables of a program source in the start condition of a UserInterruptTask, you may have to adapt the syntax of the start condition.

Task configuration - UserInterruptTasks

In the **Task configuration** tab, parameterize the error response to program errors.

You can set the following parameters:

Field/button		Meaning/information
Range limit for dynamic data (stack size)		Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode and the ShutdownTask is started.
	ExecutionFaultTask	The ExecutionFaultTask is started. All programs assigned to this task are started. If no programs are assigned, the CPU switches to **STOP** mode. The task in which the error occurred is terminated.

See also

Assigning programs to the execution levels/tasks (Page 244)

7.2.8 ShutdownTask

The **ShutdownTask** is intended for selective intervention in the transition from **RUN** to **STOP U**/**STOP** mode or for programming stop sequences with a preassigned braking ramp, e.g. selected setting of outputs, defined shutdown of axes.

The ShutdownTask is *not* called in the case of a power failure.

The **time monitoring** must be specified in the **Task configuration** for the ShutdownTask: You can configure the maximum duration for the execution of the ShutdownTask; 0 ms = no monitoring. After this time, the CPU switches to **STOP** mode.

The I/O can be accessed directly in the ShutdownTask. Access to process image and symbolic I/O variables is restricted. It is *not practical* to access the I/O by means of the process image, as the process image is no longer being updated.

For additional information, see **SIMOTION ST Structured Text**, "Access to inputs and outputs (process image, I/O variables)"

Select the **Program assignment** tab to assign the created and compiled programs to the ShutdownTask and define their execution sequence.

Note

Program errors in the **ExecutionFaultTask** and in the **ShutdownTask** switch the system to **STOP** mode immediately.

Configuring the ShutdownTask

1. Click **ShutdownTask** in the Execution Levels tree.

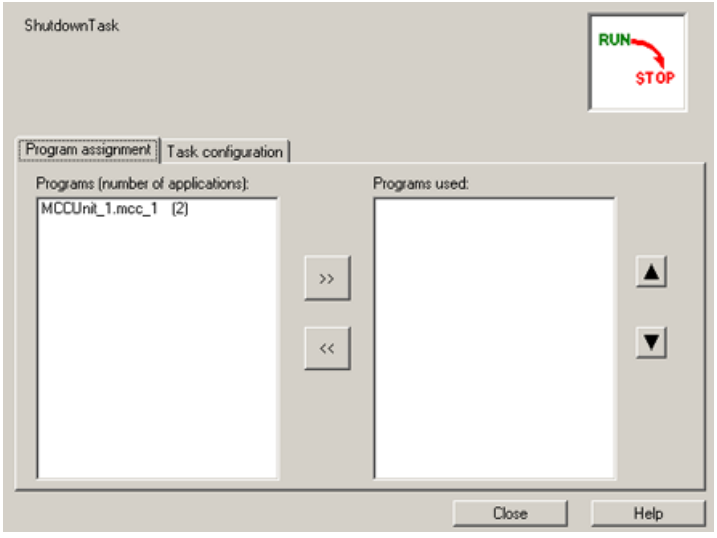


Figure 7-22 Configuration of the ShutdownTask (program assignment)

2. In the **Program assignment** tab, assign the required programs to this task and define the execution sequence.
3. Switch to the **Task configuration** tab.

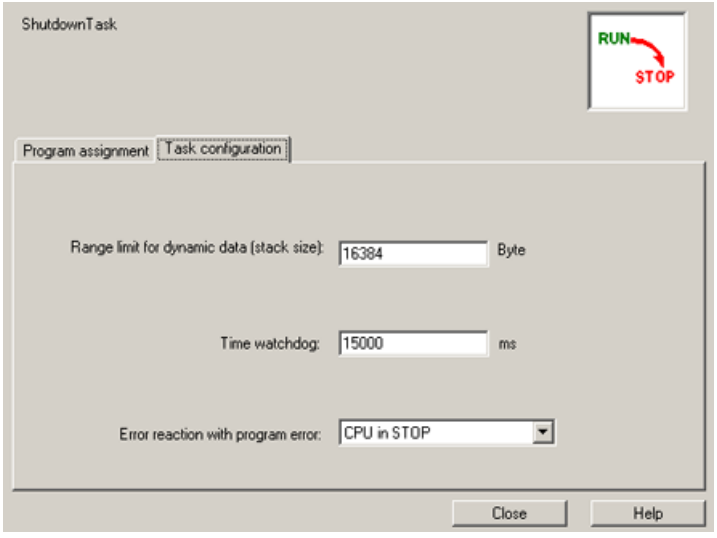


Figure 7-23 Configuration of the ShutdownTask (task configuration)

4. If required, enter the **Range limit for dynamic data (stack size)**.
5. Enter a value for the **time monitoring**.
6. Specify the **Error response to program error** (e.g. **ExecutionFaultTask**).

Task configuration - ShutdownTask

Select the **Task configuration** tab to parameterize the time monitoring.

You can set the following parameters:

Field/button		Meaning/information
Range limit for dynamic data (stack size)		Enter the local data stack size for this task in bytes. When the programs assigned to this task are executed, this size is made available for data in the stack. The guide value is 16 KB for a task. See also: • Setting the size of the local data stacks (Page 597). • "Memory requirement of the variables on the local data stack" in the SIMOTION ST Programming Manual.
Time monitoring		Specify the cycle time in ms for executing the ShutdownTask. Enter a value for time monitoring. The time monitoring is inactive if you enter 0 or no value.
Error response to program error		Select the error response for errors that occur while processing programs. Program errors are, for example, faulty operations with floating-point numbers, division by zero, and overshooting array limits.
	CPU to STOP	The CPU switches to **STOP** mode.

See also

Assigning programs to the execution levels/tasks (Page 244)

Watchdog (Page 263)

7.3 Configure execution system

Configuration of the execution system involves the following steps:

- Assignment of user programs and definition of the task properties
- Enabling of the tasks used
- Selection of the cycle clock source and setting of the system cycle clocks

7.3.1 Assigning programs to the execution levels/tasks

The programs must be assigned to execution levels. Only then are the programs executed.

After creation, you can assign the programs of an MCC source, an ST source or a LAD/FBD source to one or more tasks with SIMOTION SCOUT.

You can assign more than one program to a task.

The assigned programs are then executed in the order in which they are listed; this order can be specified and modified using SIMOTION SCOUT.

If several programs are assigned to a task, execution of the first program must be finished before the next program in the same task can be started. If, for example, the first program is in a continuous loop, the second program will never be executed.

You can also assign one program to several tasks, which are then executed independently of each other.

With the program assignment, you specify the following:

- Priority with which the programs are executed
- The execution behavior: sequential/cyclic
- The initialization behavior of program variables
  See the **SIMOTION MCC** or **SIMOTION ST** Programming Manual - "Time of variable initialization" and Influence of the compiler on variable initialization (Page 322).

Please note the following when assigning a program to one or more tasks:

- Before programs can be assigned, they must be compiled without error.
- The assignment must be made before downloading the program to the target system.
- It is possible that the program (if multiple tasks are assigned) may be called by another task while it is being executed. No measures are taken in the system to ensure data consistency.
- Once you have assigned a program to a task, it will remain assigned even in the event of recompilation.

Note

DCC tasks are assigned via the DCC editor, see the description of the DCC editor and Sequence model for DCC blocks (DCB) (Page 294).

Execution system - program assignment

You can assign the created and compiled programs to the various tasks of the execution levels in the **Program assignment** tab.

You can set the following parameters:

Field/button	Meaning/note
Programs (number of uses)	A list of all compiled programs that are available in the project is displayed here. Non-compiled programs are not displayed. The number after the program name indicates how often the program has been assigned to the different tasks of the execution levels. MCC charts and LAD/FBD programs are displayed immediately after entering as long as they have been entered as "exportable".
›› **Assign**	Use this to assign selected programs to the task. 1. Select the program in the "Programs" list. 2. Click the ›› button or drag-and-drop the program into the "Used programs" list. The program is assigned to the task and displayed in the list of used programs.
‹‹ **Remove**	Use this to remove programs assigned to a task. 1. Select the program in the "Used Programs" list. 2. Click the ‹‹ button or drag-and-drop the program into the "Programs" list. The program is removed from the task.

Field/button	Meaning/note
Used Programs	A list of all programs assigned to this task is displayed here. The sequence of the programs in the list corresponds to the execution sequence when the program is executed. The program at the top of the list is executed first.
Arrow up	Use the arrow to move the selected program up one position within the task. In this way you can determine the order of program execution within the task.
Arrow down	Use the arrow to move the selected program down one position within the task. In this way you can determine the order of program execution within the task.

Assigning programs to the tasks

1. Select the SIMOTION device in the project navigator and select **Target system > Configure execution system** in the menu or double click **EXECUTION SYSTEM**.
   The **EXECUTION SYSTEM** window opens in the working area of the workbench.

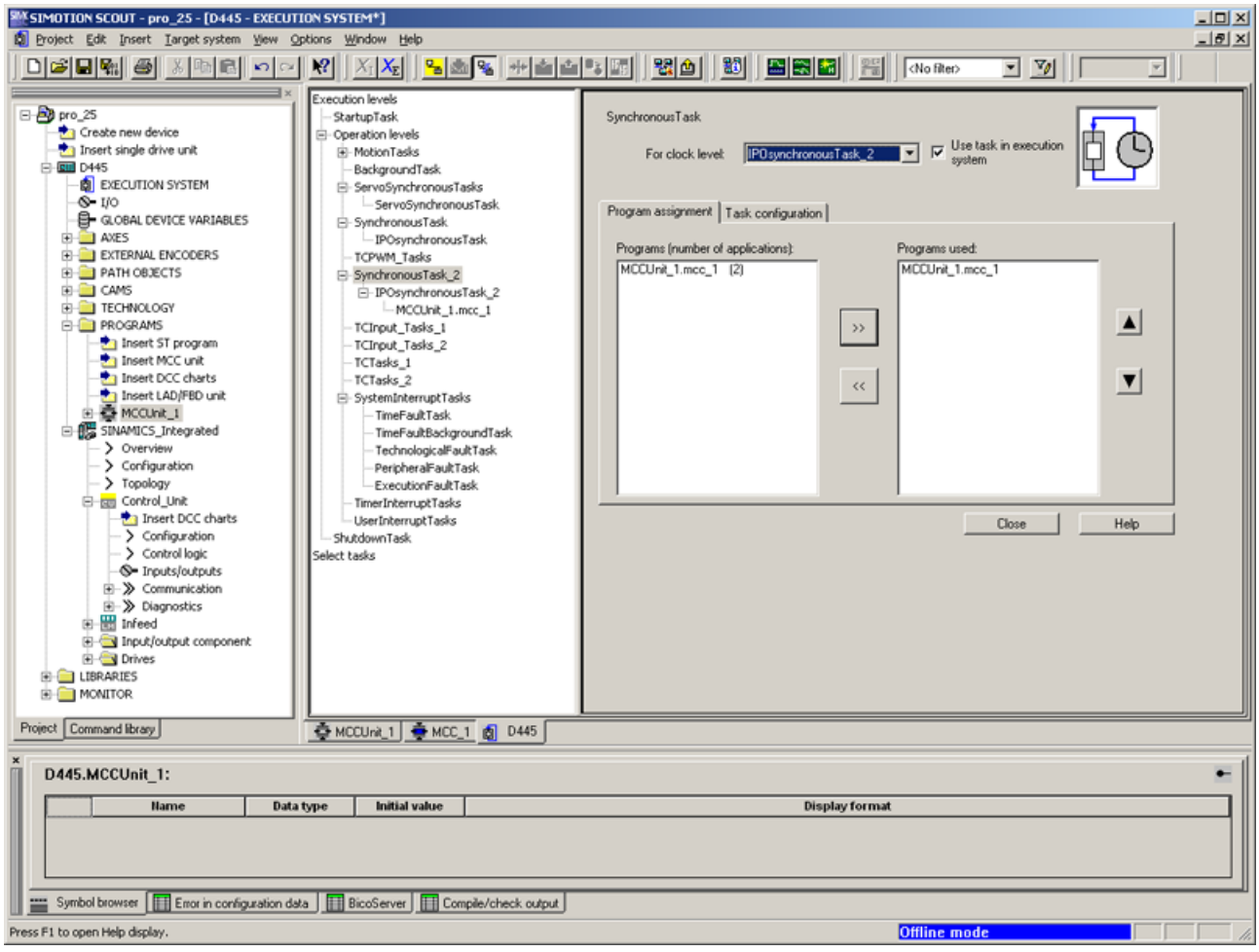


Figure 7-24 Configuration of the execution system in SCOUT

In the left-hand pane of the window, you can see the execution levels tree. It displays as entries the execution levels / tasks for which **Use task in execution level** has been clicked. The **OperationLevels** folder contains the tasks that are available in the **RUN** mode. The list below each execution level or task name shows the configured tasks and the programs assigned to them.

2. Select the task to be configured.
3. Select the **Program assignment** tab.
   The left-hand list box lists all available programs (ST programs, MCC charts and LAD/FBD with Program creation type).
4. Select the programs in the list box on the left that you want to assign to the task.

5. Click ».
   The assigned programs are listed in the right-hand list box.
   They are still listed in the left-hand list box. You can assign programs to several tasks. The number of assignments made appears in brackets.
6. Select the **Task configuration** tab and make any additional settings, such as:
   - Error response to program error
   - Time watchdogs for cyclic tasks
   - Start behavior of MotionTasks

   See the detailed description for the relevant task in Section "Description of the user program tasks (Page 212)".

After you have assigned a program to one or more tasks, you can establish the connection to the target system, download the project to the target system, and start it.

Change execution sequence

Programs are executed in the order entered. You can change this order.

1. Select the entry to be moved in the right-hand list box.
2. Click ▲ or ▼ to move the element up or down.
3. Repeat Step 2 as often as required.

Task names

The names of the MotionTasks can be changed.

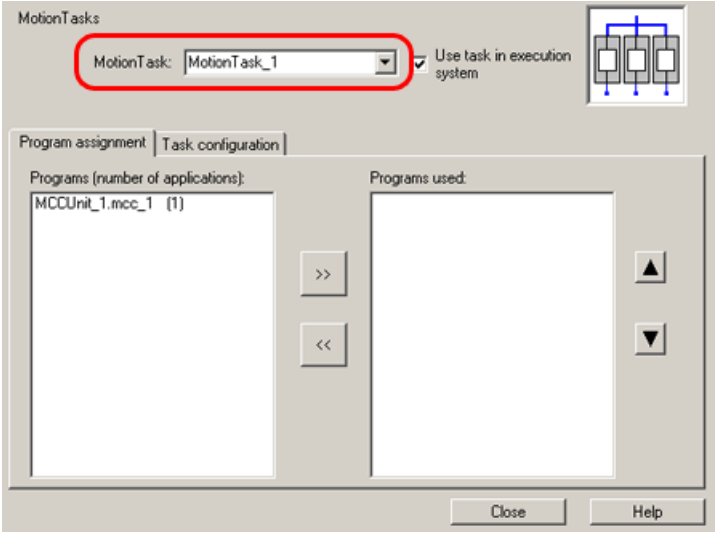


Figure 7-25 Task names in SCOUT

1. Select the desired task on the left-hand side.
2. Enter the desired new task name in the drop-down list on the right-hand side.

Task control

Various commands are available for the task control (e.g start, stop, etc.), see Section "Task control commands (Page 338)". See also Task commands in the SIMOTION MCC programming language

Information is contained there about how to use these task control commands and several examples.

Stack size

In SIMOTION SCOUT, the stack size (limit for dynamic data) of the associated task can be set on the Task configuration tab of the Task Configuration window. A default value is specified for each task.

For further information, refer to the **SIMOTION ST** Programming Manual, "Setting the size of the local data stack".

Assignment of the program to a MotionTask

In the program assignment, you can see which MotionTasks the program is assigned to via a tooltip.

See also

Task priorities (Page 205)

7.3.2 Selecting the cycle clock source

Selection of the cycle clock source is implemented in the device configuration in the **HW configuration**.

As soon as you parameterize one of the interfaces as isochronous DP/PN interface (select **"Isochronous mode"** in the Properties dialog box), the cycle clock setting is used as the bus cycle clock.

The DP/PN communication level and servo and interpolator levels are synchronized with the bus cycle clock. This setting is essential if you want to utilize the motion control functions of the **TP CAM/TP PATH/TP CAM_ext** technology package in combination with digital drives in accordance with PROFIDRIVE V3 on PROFIBUS DP/PROFINET. Drives that support and require this communication are, for example: SIMODRIVE 611U, MASTERDRIVES MOTION CONTROL, SINAMICS. Synchronization to the bus cycle clock if you must isochronously access I/O from your application.

You can still operate drives that do not support isochronous mode as drive axes on the isochronous bus. Examples of these drives are Micromaster MM4 and MASTERDRIVES VC.

If you are not using an isochronous DP/PN interface, you can set the basic system cycle clock. The servo and interpolator levels are synchronized to the basic cycle clock. You can select this setting if you are not using the **TP CAM/TP PATH/TP CAM_ext** technology package or are using it exclusively with analog drives on SIMOTION.

7.3.3 Defining system cycle clocks

Once the cycle clock source is selected, you specify the sampling times of the isochronous execution levels derived from the basic cycle clock. An additional faster servo cycle clock option and an IPO option can be configured as of V4.2, see also Servo_fast (application cycle clock for fast bus system) (Page 264).

Included here:

- **Bus cycle clock (DP cycle clock / PN cycle clock)**: refer to the table for the execution system - system cycle clocks
  The "base cycle clock" is the basic cycle clock if no isochronous DP/PN interfaces have been parameterized in HW Config. The "bus cycle clock" is based on the equidistant bus cycle and is set in HW Config. The designation changes in accordance with the setting in the **Set system cycle clocks** dialog. The basic/bus cycle clock (DP cycle clock or PN cycle clock) is used as a basis for setting further tasks.
- **Servo_fast (fast position control cycle clock (as of V4.2))**
  Inputs/outputs are updated in the Servo_fast cycle clock (optional). Servo_fast is assigned to the faster bus system (PROFINET IO) and set in HW Config.
- **IPO_fast (fast interpolator cycle clock (as of V4.2))**
  The motion control for the axes on the fast bus system is calculated in the IPO_fast cycle clock (optional).
- **Servo cycle clock (position control cycle clock) / T1(DCC) cycle clock**
  Inputs/outputs are updated in the servo cycle clock.
  The position control for the axes and the processing of the centralized and distributed I/O are carried out in the servo cycle clock. The servo cycle clock can be operated relative to the bus cycle clock at a ratio of 1:1 or 1:2. The **ServoSynchronousTask** is being processed in this clock cycle.
  For isochronous PROFINET IO, the following applies:
  - The DP master interface must run in the same cycle clock as the servo cycle clock.
  - For SIMOTION C, P and D, the bus cycle clock for the external PROFIBUS interfaces must be at least 1 ms in order for these to be operated isochronously.
  - If the PN cycle clock is scaled down (e.g. 0.5 ms), the servo cycle clock must then be the same as the PROFIBUS cycle clock (for SIMOTION D: also the same as the bus cycle clock of PROFIBUS integrated).

  Note

  The reduction ratio between the bus and servo cycle clocks must also be set as the master application cycle on the drive. This setting is necessary to enable reciprocal life-sign monitoring. For further information, refer to the drive descriptions.

- **Interpolator cycle clock (IPO cycle clock) / T2(DCC) cycle clock**
  The axis motion control is calculated in the IPO cycle clock.
  The **IPOSynchronousTask** is executed in this cycle clock. The IPO cycle clock can be reduced relative to the servo cycle clock at a ratio of 1:1 to 1:6.
- **Interpolator cycle clock 2 (IPO_2 cycle clock) / T3(DCC) cycle clock**
  The IPO cycle clock 2 is the basis for motion control of lower-priority axes.
  The **IPOSynchronousTask_2** and the **PWM Task** (TControl) are executed in this cycle clock.
  The IPO_2 cycle clock can be reduced relative to the IPO cycle clock at a ratio of 1:2 to 1:64.

- **DCCAux T4(DCC) - cycle clock**
  The DCCAux cycle clock can be reduced relative to the T3(DCC) cycle clock at a ratio of 1:2 to 1:32.
- **DCCAux_2 T5(DCC) cycle clock**
  The DCCAux_2 cycle clock can be reduced relative to the DCCAux cycle clock at a ratio of 1:2 to 1:32.
- **PWM cycle clock**: For the TControl technology package

Note

The settings of the system cycle clocks affect the system operating sequence considerably. You must therefore exercise caution when making these settings.

Set system cycle clocks

1. Call the configuration window in the main menu via **Target system > Expert > Set system cycle clocks...**, or in the context menu of the device, or in the **EXECUTION SYSTEM** context menu via **Expert > Set system cycle clocks**.

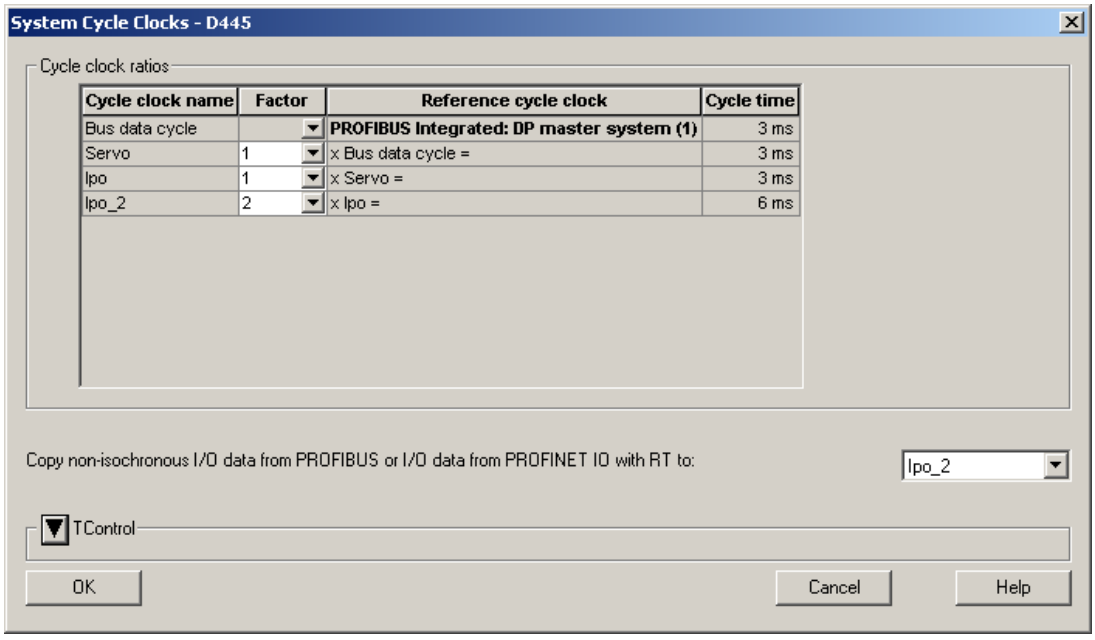


Figure 7-26 Setting of the system cycle clocks in SCOUT

2. Specify the basic/bus cycle clock (only if the isochronous mode is not configured). The bus cycle clock is specified in HW Config for the equidistant bus.
   Possible values: 1.0 ... 8.0 ms for PROFIBUS
   Possible values for PROFINET:
   - As of V4.0 of SIMOTION P and SIMOTION D4x5/D4x5-2: 0.5 ... 4.0 ms
   - As of V4.1 for SIMOTION P: 0.25 ms to 4 ms.
   - As of V4.2 for SIMOTION D: 0.25 ms to 4 ms.
   - As of V44 for SIMOTION D455-2 DP/PN: 0.125 ms to 4 ms.

   The cycle time for PROFIBUS must be an integer multiple of 0.125 ms, or of 0.25 ms in the case of SIMOTION C.
   The cycle time for PROFINET must be an integer multiple of 0.125 ms. Change the value in **HW Config** if this is not the case.
3. Specify the ratio between the cycle clocks.

Note

If the basic/bus cycle clock has been set too small for a large hardware configuration, this can result in the CPU not switching to the RUN mode. In this case, note the entries in the diagnostics buffer for timeouts.

Set system cycle clocks with two servo cycle clocks (Servo_fast)

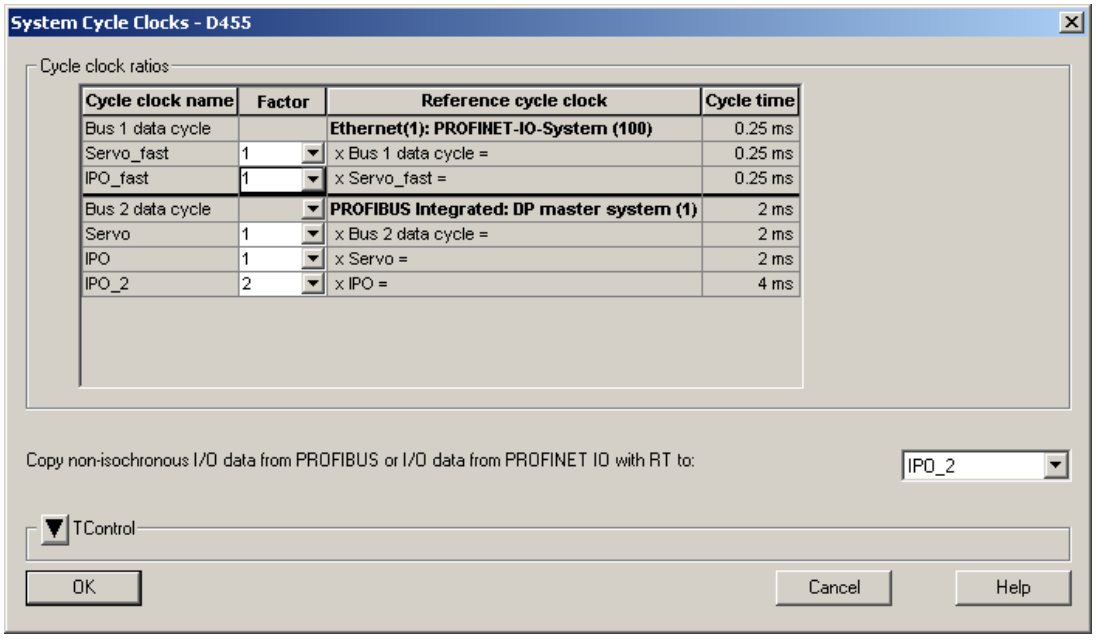


Figure 7-27 Servo and Servo_fast system cycle clocks

A description of how to configure a second servo cycle clock can be found in the *Communication* Function Manual under Servo_fast, scaling down of cycle clocks to the servo at the PROFINET interface.

System cycle clocks for DCC

The entries for DCC are displayed as soon as a chart has been added under programs. DCC charts are started automatically when the appropriate task is started.

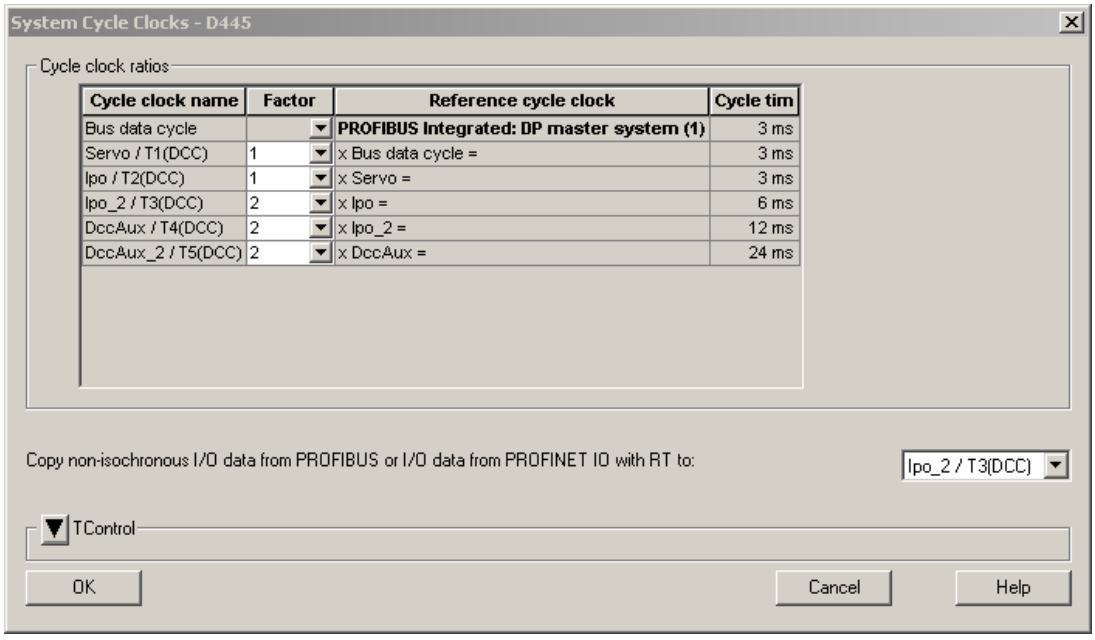


Figure 7-28 Setting of the system cycle clocks in SCOUT with DCC

Setting the system cycle clocks for TControl

1. You can use the **TControl** button in the **System Cycle Clocks** configuration window to display the settings for the TControl system tasks.
2. You can use the **Use system tasks for TControl** checkbox to activate or deactivate the TControl system tasks.
   If **Use system tasks for TControl** is activated, the special tasks for the temperature channels are displayed in the execution system.
   If you have not configured a temperature channel, you should deactivate the system tasks for TControl as they require unnecessary computation time.
3. Furthermore, you can specify here the cycle clock ratios of the TControl tasks.
   - The **PWM** cycle clock must be an integer multiple of the position control cycle clock.
   - The cycle clock of **InputTask_1/2** depends on the PWM cycle clock.
   - The cycle clock of **PostTask_1/2** depends on the cycle clock of **InputTask_1/2**.

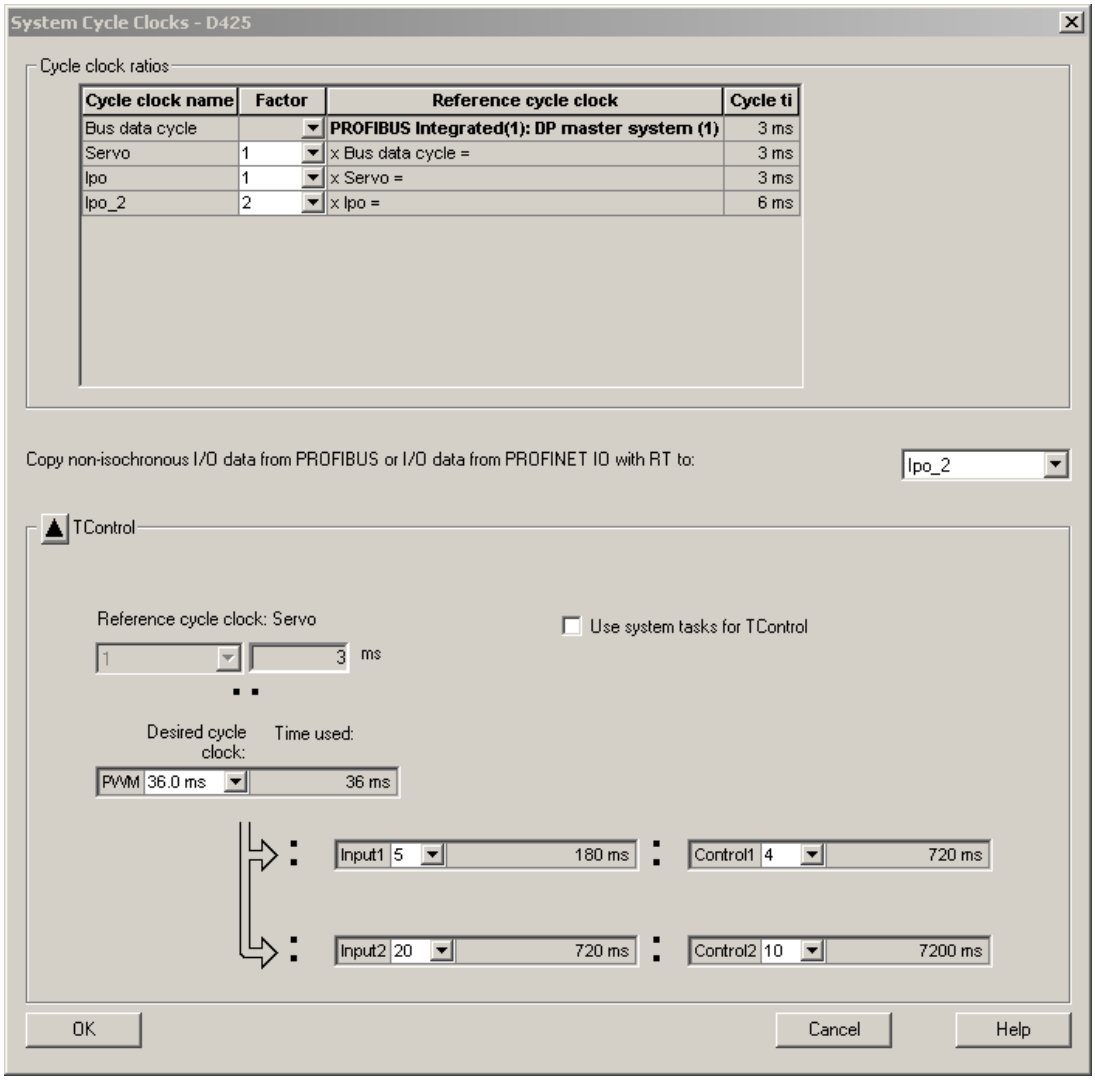


Figure 7-29 Setting of the system cycle clocks for TControl in SCOUT

Execution system - system cycle clocks

You can set the following parameters:

Field/button		Meaning/information
Cycle clock ratios		
	Basic cycle clock or **bus cycle clock**	The base cycle clock is the basic cycle clock if no isochronous DP/PN interfaces have been parameterized in HW Config. The "bus cycle clock" cannot be set in this dialog box if the "Equidistant bus cycle" setting is activated. The "bus cycle clock" is set in HW Config. This means the equidistant bus cycle forms the basic cycle clock for the system cycle clocks. The basic/bus cycle clock (DP cycle clock or PN cycle clock) is used as a basis for setting further tasks.
	Servo_fast	You can set the Servo_fast cycle clock here (optional). The reference cycle clock is the isochronous PROFINET IO cycle clock (for the faster bus system) set in HW Config.
	IPO_fast	You can set the IPO_fast cycle clock here. The reference cycle clock is the Servo_fast cycle clock.
	Servo / T1(DCC)	You can set an integer ratio between the servo cycle clock and the bus/Servo_fast cycle clock here. Among others, the position control of the axes is calculated in this cycle clock. Normally the factor 1 should be used. If you set the factor 2, although the dynamic performance of the controller will deteriorate, more computing time will be available for processing other tasks. The reduction ratio between the bus cycle and the servo cycle clock must also be set as the "master application cycle" on the drive. This setting is necessary to enable reciprocal life-sign monitoring. For further information, refer to the drive descriptions. If isochronous mode is not configured, the default setting for the basic time to servo cycle clock ratio is 1:1. It cannot be changed then! PROFIBUS DP: The servo cycle clock can be operated at a ratio of 1:1, 1:2, 1:3, and 1:4. PROFINET IO: The servo cycle clock can be operated at a ratio of 1:1, 1:2, 1:3, 1:4, 1:6, 1:8, 1:10, 1:12, 1:14, and 1:16.
	Ipo / T2(DCC)	You can set an integral ratio between the IPO cycle clock and servo cycle clock here. As standard, the motion control of the axes will be calculated in the interpolator cycle clock. The interpolator cycle clock factor is used to determine how fast the drive setpoints are calculated.
	Ipo2 / T3(DCC)	You can set an integral ratio between the IPO_2 cycle clock and IPO cycle clock here. The interpolator cycle clock 2 is used for the motion control of low-priority axes. The factor is used to determine how fast the setpoints of the low-priority drives are calculated.
	DCCAux T4(DCC)	You can set an integer ratio between the DCCAux DCC cycle clock and T3(DCC) DCC cycle clock here.
	DCCAux_2 T5(DCC)	You can set an integral ratio between the DCCAux_2 DCC cycle clock and DCCAux DCC bus cycle clock here.
TControl		Click the arrow to open the configuration of the system tasks for the TP TControl. There, you can set the ratio of the cycle clocks for the task of the temperature channels to the servo cycle clock.

Field/button		Meaning/information
	Use system tasks for TControl	Activate the checkbox if the special tasks for the temperature channels in the execution system should be displayed. If you have not configured a temperature channel, you should deactivate the system tasks for TControl as they require unnecessary computation time.
	Servo Reference Clock Cycle	The servo cycle clock is displayed here for information purposes. To change this, you must select the value further up in the dialog box. All other cycle clocks of the temperature channel are a multiple of the servo cycle clock.
	PWM (Pulse Width Modulation)	Cycle clock which is used for the actuating signal output of the temperature channel. Specifies the basic cycle clock for further cycle clocks. Select the cycle clock in ms or in Hz. If you select Hz, the ratios to the input cycle clock and the control cycle clock are specified automatically.
	Input1/2	Cycle clock which is used for the actual value measurement of the temperature channel. Select the ratio of this cycle clock to the PWM cycle clock. The duration is displayed in the grayed-out field.
	Control1/2	Cycle clock which is used for the closed-loop control of the temperature channel. Select the ratio of this cycle clock to the input cycle clock. The duration is displayed in the grayed-out field.

Copying asynchronous cyclic I/O data

As of V4.2, you can choose when the asynchronous cyclic I/O data is to be copied. This means you can now choose whether data is copied in the IPO (with DCC T2 (DCC)) or IPO_2 (with DCC T3 (DCC)).

- To do this, select the relevant cycle clock from the drop down list box **Copy non-equidistant I/O data from PROFIBUS or I/O data from PROFINET IO with RT to**.

The inputs are always copied at the start of the first task on a level. The outputs are always copied at the end of the last task on a level. If there is no DCC or user program task, the BackgroundTask serves as both the first and the last task on the level concerned.

Note

Copied are all non-equidistant data on the **PROFIBUS DP**, **PROFINET IO**, and**I/O bus**.

7.3.4 Assigning system cycle clocks to technology objects

Assigning system cycle clocks

You can make better use of the performance reserves of the SIMOTION CPU by assigning priorities to your technology objects and specifically assigning them to system cycle clocks. The definition of lower-priority tasks allows you to gain performance reserves for higher-priority tasks.

Consequently, a higher time level can be used for the TOs, particularly for uniform motions without large accelerations/decelerations. (e.g. call of the TO in the interpolator cycle clock 2).

Change the standard setting if you detect:

- A job processing that lasts too long
- Too great a utilization of the technology in the CPU

Assign the interpolator cycle clock 2 to the axis and external encoder technology objects with lower-priority tasks, and assign the IPO cycle clock or the interpolator cycle clock 2 to the output cam and the measuring input technology objects.

Assign the interpolator cycle clock or the servo cycle clock to the technology objects with high-priority tasks.

You can assign the following cycle clocks to the technology objects:

Motion control task	**High priority**	**...**	**Lower priority**
Technology object	Servo cycle clock	Interpolator cycle clock	Interpolator cycle clock 2
Drive axis	(x)	Standard	X
Position axis	(x)	Standard	X
Following axis	(x)	Standard	X
External encoder	(x)	Standard	X
Output cam	X	X	X
Cam track	X	X	X
Measuring input	X	X	X

The system cycle clocks for output cams and measuring inputs, as well as for axes, are set in SIMOTION SCOUT in the **Configuration** dialog.

As of V4.1 in exceptional cases the IPO share can also be calculated in the servo, in this regard see **Motion control / interpolator** in the **TO axis** manual.

See also

Second position control cycle clock (Servo_fast) (Page 264)

7.3.5 Task runtimes

You can use the task runtimes to check whether the computer performance of the system is sufficient for the demands of the application.

The task runtimes are displayed in the **Taskruntime** and **effectiveTaskruntime** device variables.

A distinction is made between:

- The runtimes of the individual tasks within a level (**Taskruntime**)
  The **Taskruntime** displays the sum of the net runtimes of the task (without the interrupt times).
- The runtime of the level (**effectiveTaskruntime**)
  The **effectiveTaskruntime** displays the effective runtime of a level (including the interrupt times). This is the time from the start of the level until the end of the last task in the level.

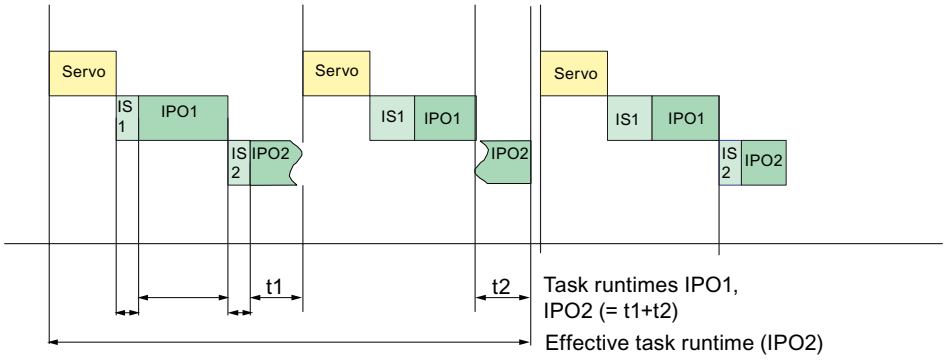


Figure 7-30 Representation of the Taskruntime and effectiveTaskruntime (cycle clock ratio IPO1 : IPO2 = 2)

Legend:

Servo: Servo cycle clock

IS1: IPOSynchronousTask

IS2: IPOSynchronousTask_2

IPO1: IPOTask

IPO2: IPOTask2

The following table shows the tasks and the levels for which a Taskruntime (tasks) and an effective Taskruntime (levels) can be determined.

Tasks (Taskruntime)	Levels (effective Taskruntime)
servodcc	Servo / T1(DCC)
servo	
servosynchronous	
servo_fast	Servo_fast
Ipodcc	Ipo / T2(DCC)
ipo	
iposynchronous	
ipo_fast	IPO_fast
ipodcc_2	Ipo_2 / T3(DCC)
ipo_2	
Iposynchronous_2	
dccaux	DCCAux T4(DCC)

Tasks (Taskruntime)	Levels (effective Taskruntime)
dccaux_2	DCCAux_2 T5(DCC)
background	Background[1]

[1] With "Background" level: Difference between 2 start times of the BackgroundTask

Note

To determine the task runtimes, the system variable **taskRuntimeMonitoring** must be set to **enable**.

The task runtimes are also shown in the SCOUT device diagnostics.

Note

Even when the module is in the **Stop** operating state, system tasks use computing time.

See also

Functions for runtime measurement of tasks - overview (Page 349)

Optimizing the execution system (Page 600)

Efficient programming - overview (Page 598)

7.3.6 Timeouts and level overflows

Timeouts and/or level overflows may occur when executing computing processes. To monitor task runtimes, you can use the device diagnostics in SCOUT (ONLINE only). The evaluation of system variables is described in **Monitoring of timeouts and level overflows** (see below).

You have the possibility of configuring various responses for the system response to overflows of certain execution levels.

Timeout

You can configure maximum execution cycles for the execution levels **BackgroundTask**, **TimerInterruptTask**, **SynchronousTasks** and **ShutdownTask**.

If the set maximum duration is exceeded, the associated **SystemInterruptTask** (**TimeFaultTask** or **BackgroundFaultTask**) or a standard function (e.g. CPU to **STOP**) can be called up.

Level overflow

A toleration of level overflows can be set for the execution levels **IPO1** and **IPO_2**.

A level overflow occurs when a level (**IPOTask/IPOSynchronousTask** or **IPOTask2/ IPOSynchronousTask_2**) has not been executed within the set system cycle clock, IPO or IPO_2.

Note

A cyclic user task (IpoSync, Ipo2Sync) must be finished within its cycle.

A task overflow caused by a failed start attempt of a lower priority task produces an entry in the diagnostic buffer.

If the clock cycles be set too low so that level overflows continuously occur, the CPU will no longer function.

With a set tolerance, the next IPO cycle clock is used in order to:

- Complete the computing process of the previous cycle clock without triggering the set error response
- Start the computing process of the current cycle clock.

If another level overflow occurs in the next level cycle (if no tolerance is set or the number has been exceeded), the CPU is set to **STOP** in order to prevent a blockade of the entire system. This results in an entry in the diagnostic buffer and the error response (CPU to **STOP**) with starting lockout.

You can set a maximum of 5 level overflows to be tolerated in the task configuration in the execution system.

Examples

The following are examples of the execution relationships for level overflows:

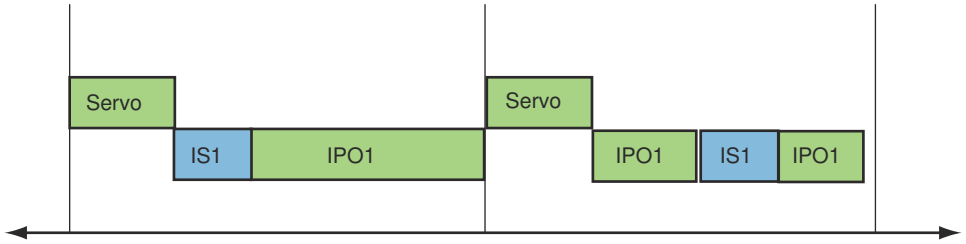
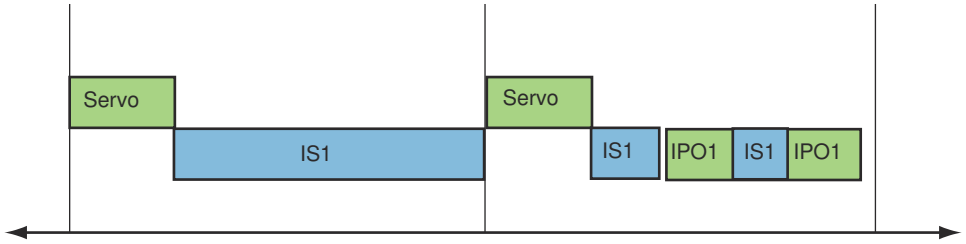


Figure 7-31 IPO overflow example



Servo: Servo cycle clock
IS1: IPOSynchronousTask
IPO1: IPOTask

Figure 7-32 Overflow of the IPOSynchronousTask example

High-performance programming

If level overflows occur, but it is not possible to increase the DP/servo cycle clock, the following measures can be taken:

- In task configuration for IPOSynchronousTask
  Ratio of IPOSynchronousTask : IPO cycle clock = 75%
  Tolerate two IPO overflows
- Use optimized PROFIBUS
  User-defined profile: HSA=2 (highest PROFIBUS address)
  with, for example, two nodes on the PROFIBUS, RetryLimit = 1, switch off cyclic distribution of the bus parameters (then, however, a PG can no longer be connected to the isochronous PROFIBUS)
- Change ratio BackgroundTask : MotionTasks
  If, for example, focus is placed on the MotionTask, an event in the BackgroundTask results in a MotionTask being quickly started and handled with few interruptions.
- Read system variables only once and temporarily store them in a local variable for later use

For additional information on high-performance programming, refer to Optimizing access to inputs and outputs (Page 599) .

Monitoring of timeouts and level overflows

The system variables **Taskruntime** and **effectiveTaskruntime** can be used to determine whether a level overflow or a timeout has occurred.

- If the **Taskruntime** in the IPO/IPO_2 cycle clock is *the same* as the **effectiveTaskruntime** in the IPO/IPO_2 cycle clock, then the task has *not* been interrupted by a higher-priority task.
- If the **Taskruntime** in the IPO/IPO_2 cycle clock is *not the same* as the **effectiveTaskruntime** in the IPO/IPO_2 cycle clock, then the task has been interrupted by a higher-priority task.
  Recommendation: With a ratio of servo : IPO : IPO_2 > 1, enter a percentage value of the IPO cycle clock for the duration which is as large as possible in the task configuration of the **SynchronousTasks**.
- If the **effectiveTaskruntime** of the level which has overflowed is *less* than the set cycle clock of the level, a *timeout* has occurred.
  You can prevent this by adjusting the monitoring time of the levels to the values of the system variable **effectiveTaskruntime**.
- If the **effectiveTaskruntime** of the level which has overflowed is *very close to or greater* than the set cycle clock of the level, a *level overflow* has occurred.
  You can prevent this by adjusting the system cycle clocks, or you can tolerate these overflows.

See also

SynchronousTasks (Page 224)

Task runtimes (Page 258)

7.3.7 Information on starting a task: TaskStartInfo (TSI)

Every time a task is started, important information is stored in the **TaskStartInfo**, e.g.:

- The starting time of the task
- For the **TechnologicalFaultTask**: The triggering instance of the technology object and the alarm number
- For the **TimeFaultTask**: The TaskId of the **TimerInterruptTask** which caused the timeout error
- for **the ExecutionFaultTask**: The triggering event and the TaskId of the task in which the event occurred.

Within a task, you can query the relevant **TaskStartInfo** of this task, see Using Taskstartinfo (Page 160).

7.3.8 Watchdog

The maximum duration for the execution of a task (cycle time) can be configured. If this time is exceeded, the associated SystemInterruptTask (**TimeFaultTask** or **TimeFaultBackgroundTask**) or the response CPU to **STOP** mode can be called.

The cycle time monitoring can be activated for the following tasks:

- BackgroundTask
- TimerInterruptTasks
- SynchronousTasks
- ShutdownTask

Configure cycle watchdog

- Switch to the **Task configuration** tab in the Configuration window for the task.

For BackgroundTask, TimerInterruptTask, ShutdownTask:

- Enter the maximum execution time in the **Time monitoring** field (0 ms = no monitoring).

For IPOsynchronousTask:

- Choose the ratio between the maximum execution period and the IPO cycle clock.

For tasks other than ShutdownTask:

- Select as error response to timeout either the associated **SystemInterruptTask** or the CPU to **STOP** mode response.

Note

For time-triggered tasks: If the task is restarted before it has been executed, there is a timeout error. The SystemInterruptTask is started or the CPU goes to **STOP** mode.

See also

SystemInterruptTasks (Page 232)

7.4 Second position control cycle clock (Servo_fast)

7.4.1 Servo_fast (application cycle clock for fast bus system)

Description

Synchronization (data copying, cyclic life-sign, time stamp for output cam and measuring input) with all the synchronous I/O (drive, external encoder, measuring input input, output of output cam, digital and analog inputs/outputs) occurs in the position control cycle clock. Further synchronous cycle clocks (IPO, IPO_2, DccAux, DccAux_2) are based on this cycle clock.

With only one servo, there is only one common application cycle clock available for all the bus systems. This application cycle clock may not be shorter than the cycle clock of the slowest bus system (example: PROFIBUS with 1 ms). Although the faster bus system (e.g. PROFINET IO with 250 µs send clock) is able to transfer the data using a faster cycle clock, the signals are only evaluated in the slow application cycle clock.

Second position control cycle clock (Servo_fast) for the fast bus system

As of V4.2, you can operate synchronous devices (drive, measuring input or output of output cam, I/Os) on the fast bus system (PROFINET) via Servo_fast. Servo_fast always finishes during the first send clock of the fast bus system with this type of arrangement. This makes it possible to operate two bus systems in different application cycle clocks. An assigned position control cycle clock and IPO cycle clock are available for each of the two application cycle clocks.

A description of how to configure a second position control cycle clock can be found in the *Communication* Function Manual under Servo_fast, scaling down of cycle clocks to the servo at the PROFINET interface.

A second position control cycle clock is only necessary in applications with special requirements, e.g.:

- For fast I/O processing via PROFINET IO (for particularly short sampling times and response times)
- If, in addition to the electric axes (e.g. servo drives), hydraulic axes with particularly high-performance pressure/position control are implemented
- If the electric axes have to be split into two performance classes (fast servo - slow servo)

Note

The following SIMOTION modules support the second position control cycle clock:

- V4.2 and higher SIMOTION D445-2 DP/PN and SIMOTION D455-2 DP/PN.
- V4.3 and higher SIMOTION D435-2 DP/PN.

If the axes have to be split into two performance classes, a split on the IPO level (IPO, IPO_2) is usually sufficient; a split on the servo level (Servo, Servo_fast) is only required in exceptional circumstances.

- The reference variable calculation/motion profile calculation of the axes is performed on the IPO level.
- The position control and monitoring of the axes is performed on the servo level.

Please also note that an additional Servo_fast/IPO_fast increases the overall load on the SIMOTION runtime system.

In addition, with DSC (Dynamic Servo Control) for servo axes, the dynamically effective part of the position controller in the drive is performed in the frequency of the speed control loop (i.e. usually with 125 µs in the case of SINAMICS S120). If symbolic assignment with automatic message frame determination is used, DSC is activated automatically.

Servo_fast and IPO_fast

This means the additional fast application cycle clocks can also be selected for the technology objects:

Expansion to include Servo_fast and IPO_fast	Expansion to include IPO_fast
TO axis	TO AdditionObjectType
TO external encoder	TO path object
TO synchronous operation	TO fixed gear
TO measuring input	TO formula Object
TO output cam	TO sensor
TO cam track	TO controller object

See also

Cycle clock model based on a single position control cycle clock (Page 265)

7.4.2 Cycle clock model based on a single position control cycle clock

Description

The use of Servo_fast is optional and is only intended for certain applications. See also Servo_fast (application cycle clock for fast bus system) (Page 264). An application cycle clock is sufficient for standard applications.

The graphic below shows an arrangement based on a single position control cycle clock.

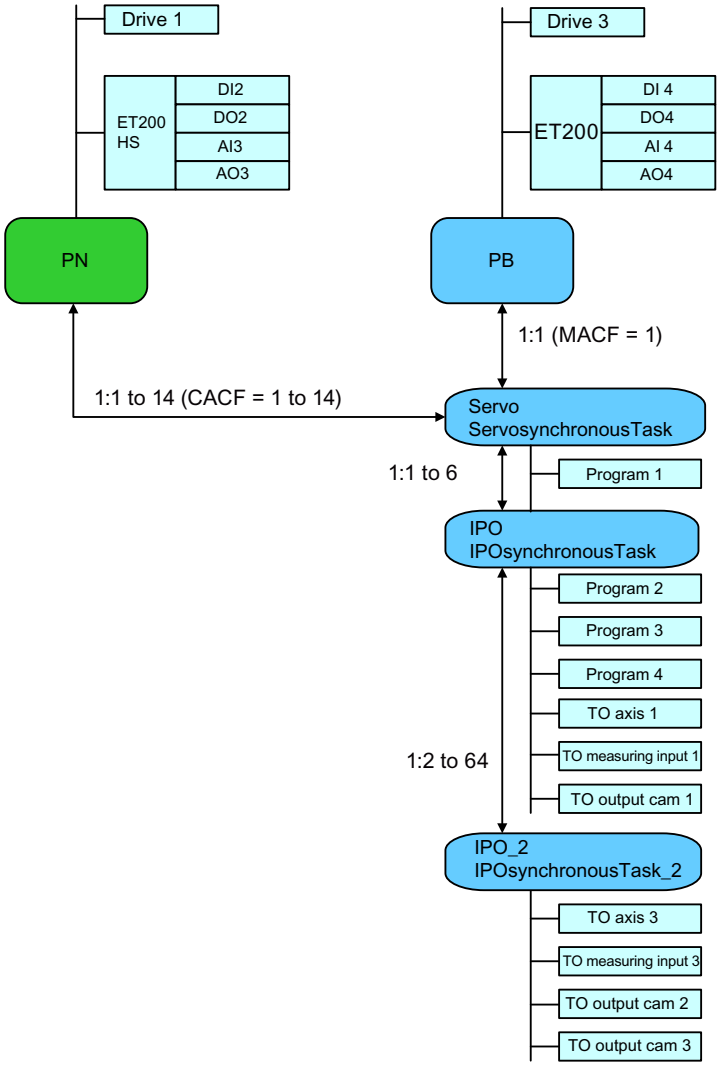


Figure 7-33 I/O with a single position control cycle clock

- MACF: The Master Application Cycle Factor describes the reduction ratio of the PROFIBUS DP send clock to the cycle clock of a higher-level control.
- CACF: The Controller Application Cycle Factor describes the reduction ratio of the PROFINET IO send clock to the cycle clock of a higher-level control.

7.4.3 Cycle clock model based on two position control cycle clocks

Description

The second servo cycle clock enables you to operate two bus systems in different application cycle clocks. An assigned servo cycle clock and IPO cycle clock are available for each of the two application cycle clocks. This enables you to divide your application into slow and fast sections (Servo_fast and IPO_fast).

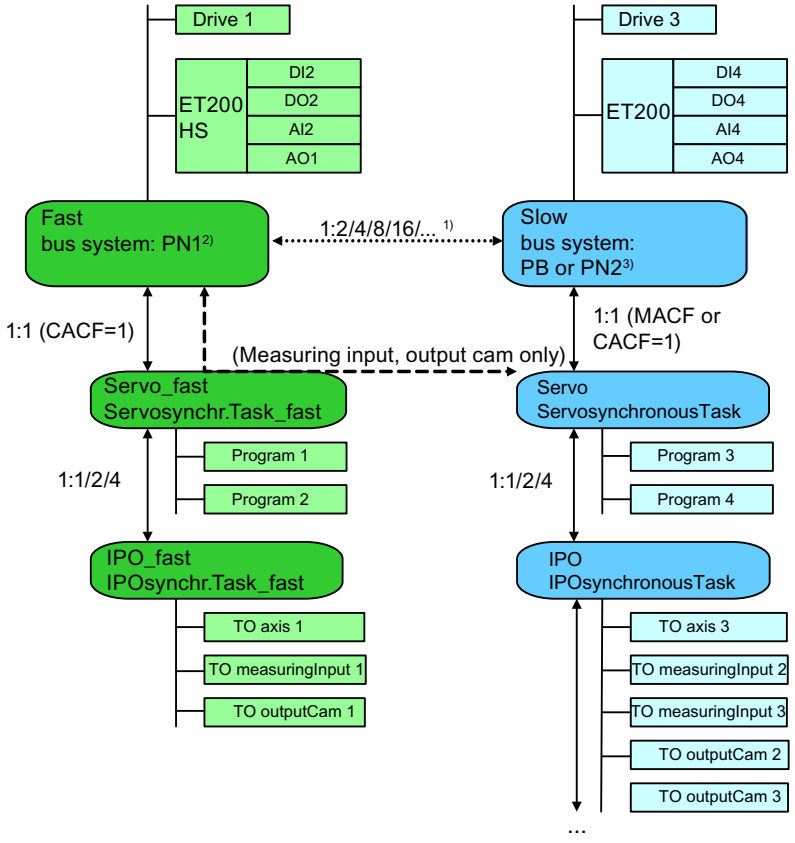
The basic principle is as follows:

- The TOs and I/Os on the fast bus system are processed isochronously in the fast Servo_fast/ IPO_fast.
- The TOs and I/Os on the slow bus system are processed isochronously in the slow Servo/IPO/ IPO_2.

Exception:

- The TO output cam and measuring input in the slow Servo/IPO/IPO_2 can also use the I/O of the fast bus system.

The graphic below shows an arrangement involving two servo cycle clocks.



1) Factor = 2 only for PN1 send cycle clock > 250 µs (for < V4.4 only factor 2, 4, 8 and 16 for all PN1 send cycle clocks)
2) PN1: Onboard PROFINET IO interface for D435-2 DP/PN, D445-2 DP/PN and D455-2 DP/PN
3) PN2: Optional second PROFINET IO interface with CBE30-2 (as of V4.3)

Figure 7-34 I/O with two servo cycle clocks

Generally, a PROFINET send cycle of max. 4 ms and a PROFIBUS bus cycle of max. 8 ms can be set.

For information on CACF and MACF, see also Cycle clock model based on a single position control cycle clock (Page 265).

7.4.4 Cycle clock ratios and conditions with two servo cycle clocks

Description

For isochronous use, the application cycle is set in HW Config on the I/O module:

- The MACF (master application cycle) is set for PROFIBUS DP slaves. Only MACF=1 is supported if you are also using PROFINET IO. All PROFIBUS devices run in the slow servo cycle clock.
- The CACF is set for PROFINET IO devices. As of V4.2, you can also select the servo cycle clock (Servo_fast) on I/O devices. As of V4.3, a second PROFINET IO interface can be connected in D4x5-2 DP/PN Control Units. This allows the integrated PROFINET interface (X150) to be operated in the Servo_fast clock cycle and the second PROFINET IO interface (CBE30-2) in the Servo clock cycle.

The following settings are possible:

Product version	Property	Application cycle of the devices on		
		PROFINET IO Integrated (X150)	PROFINET IO (CBE30-2)	PROFIBUS DP
Before V4.2	1 servo cycle clock	--	--	Servo
As of V4.2 in addition	2 servo cycle clocks	Servo_fast	--	Servo
As of V4.3 in addition	2 servo cycle clocks	Servo_fast	Servo	Servo

Application cycle on the I/O device

The application cycle set in HW Config on the I/O device has the following effects on the isochronous use of technology objects and user programs:

- Standard I/O are only used in a precisely isochronous manner in the relevant servo. An "IPO-synchronous" process image of the outputs can be output in the 1st, 2nd, 3rd, etc. servo cycle clock within an IPO (with scaling down of the cycle clocks, servo to IPO) (depending on the servo + IPO runtimes). This means the interval between 2 outputs may fluctuate.
- Drive and encoder telegrams are synchronized in the relevant servo between the controller and device. They can be used isochronously by the TO axis in the associated IPO.
- Measuring inputs and output cam outputs with time stamps can also be used synchronously in a slower cycle clock.

Application cycle clock set	Cycle clocks for the isochronous use of		
	Standard I/O	Drive, external encoder	Measuring input, output cam output
Servo_fast	Servo_fast	Servo_fast IPO_fast	Servo_fast IPO_fast Servo IPO IPO_2
Servo	Servo	Servo IPO IPO_2	Servo IPO IPO_2

Factors and reference cycle clocks with two servo cycle clocks

The table below contains the factors and reference cycle clocks that are possible with 2 servos:

Task name	Factors which can be set	Reference cycle clock
Servo_fast	1	Send cycle clock of the onboard PROFINET IO interface (X150)
IPO_fast	1, 2, 4	Servo_fast
Servo	1	PROFIBUS DP bus cycle clock and send cycle clock of the optional second PROFINET IO interface (CBE30-2)
IPO	1, 2, 4	Servo
IPO_2	2, 3, 4, 5, ..., 64	IPO
DccAux	2, 3, 4, 5, ..., 32	IPO_2
DccAux_2	2, 3, 4, 5, ..., 32	DccAux

Possible cycle clock settings for equidistant cycle clocks

The following conditions must also be satisfied when using two servo cycle clocks:

- PROFIBUS DP bus cycle clock = N x send cycle clock of the onboard PROFINET IO interface X150
  N = 2, 4, 8, 16, ... (N = 2 only for send cycle clock > 250 µs)
  (for < V4.4: N = 2, 4, 8, 16 for all send cycle clocks)
- Send cycle clock of an optional second PROFINET IO interface (CBE30-2) = PROFIBUS DP bus cycle clock
- IPO ≥ IPO_fast
- Servo_fast and IPO_fast are always assigned to the onboard PROFINET IO interface X150
- The onboard PROFINET interface can only be operated as sync master (consequently with distributed synchronous operation via PROFINET, the D4x5-2 cannot participate as sync slave when Servo_fast/IPO_fast is used)
  If the onboard PROFINET interface is configured as sync slave, the faulty configuration is indicated via a diagnostic buffer entry and the controller goes into starting inhibit.

7.4.5 Priorities and sequences of synchronous tasks

7.4.5.1 Cycle clock division when there are two position control cycle clocks

Description

The following rules apply in terms of task priorities:

- Task levels with a shorter cycle time have a higher priority than task levels with a longer cycle time.
- If the cycle time is the same, the servo level has a higher priority than the IPO level.
- If the cycle time is the same, the fast IPO level (IPO_fast) has a higher priority than the slow IPO level (IPO).

There are two synchronous task sequences dependent upon the cycle clock settings. The following graphic shows the two options for the priority and sequence of the task levels.

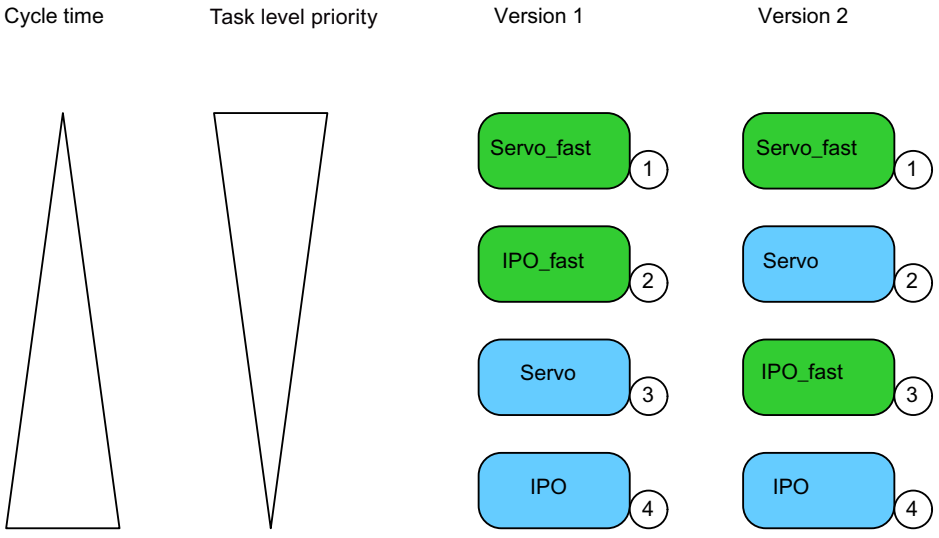


Figure 7-35 Dividing cycle clocks when there are two position control cycle clocks

Rules for cycle clock division

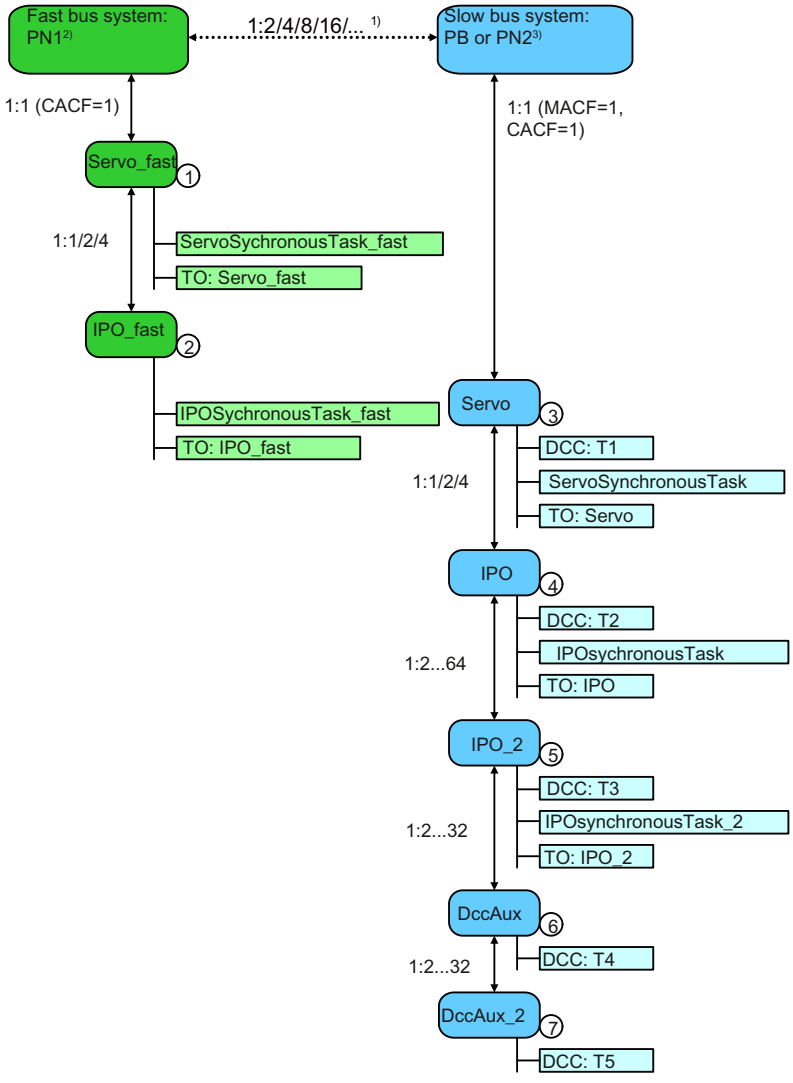
- If the setting is made for scaling down of cycle clocks between PROFINET IO and PROFIBUS DP as well as between Servo_fast and IPO_fast, the user has a choice between version 1 (Servo_fast – IPO_fast – Servo – IPO) and version 2 (Servo_fast – Servo – IPO_fast – IPO).
- If IPO_fast has a shorter runtime than the servo, the priorities for version 1 are set within the execution level system.
- A very fast **terminal-terminal response** requires a very short cycle clock for Servo_fast. If there is a strong possibility that IPO_fast may not be finished in good time before Servo due to a high system load, then IPO_fast must be set to the same speed or a slower speed than Servo. The priorities for version 2 are then set within the execution level system.
- The IPO_fast cycle clock is not longer than the IPO cycle clock.

- The new Servo_fast and IPO_fast cycle clocks must not overrun.
- In the event of a level overflow, the system goes into STOP mode. The starting lockout is set and an appropriate entry is made in the diagnostics buffer. Only after a ramp-up (power on/off) or download can the system return to RUN mode.

7.4.5.2 Cycle clock division for the 2nd servo, version 1

Task assignment, version 1 (Servo_fast – IPO_fast – Servo – IPO)

The graphic below shows version 1 of cycle clock assignment.



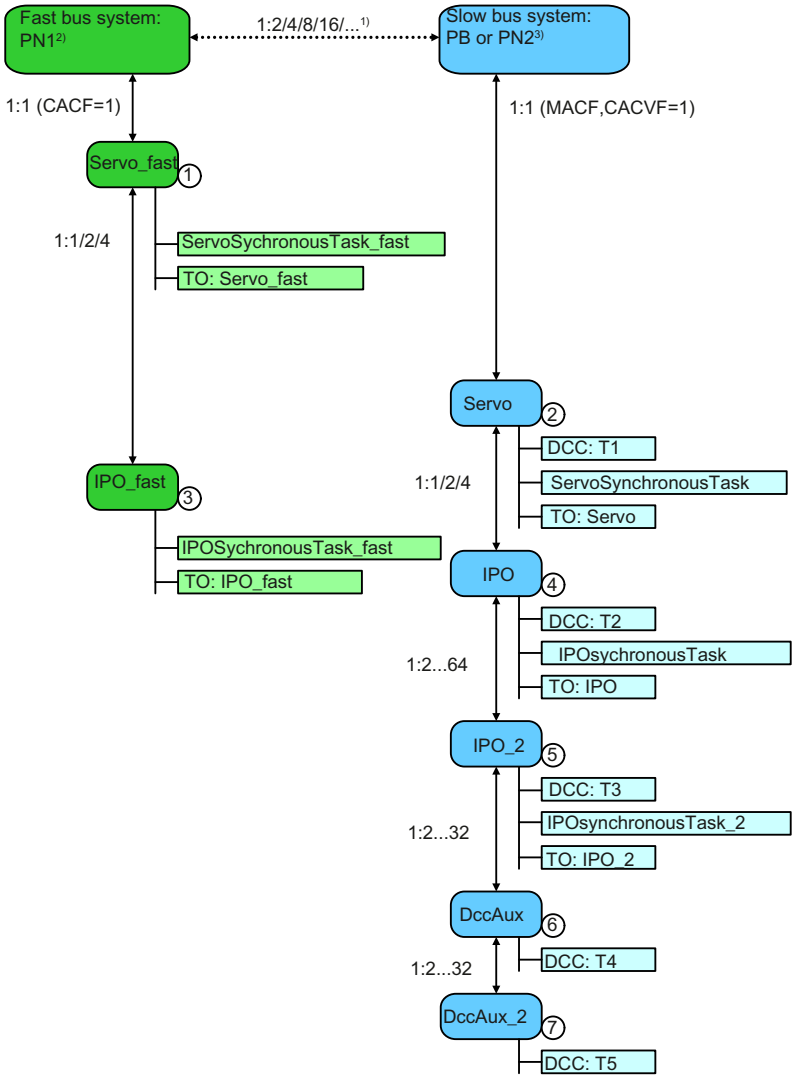
[1] Factor = 2 only for PN1 send cycle clock > 250 µs (for < V4.4 only factor 2, 4, 8 and 16 for all PN1 send cycle clocks)
[2] PN1: Onboard PROFINET IO interface for D435-2 DP/PN, D445-2 DP/PN and D455-2 DP/PN
[3] PN2: Optional second PROFINET IO interface with CBE30-2 (as of V4.3)

Figure 7-36 Cycle clock division, version 1

7.4.5.3 Cycle clock division for the 2nd servo, version 2

Task assignment, version 2 (Servo_fast– Servo– IPO_fast – IPO)

The graphic below shows version 2 of cycle clock assignment.



[1] Factor = 2 only for PN1 send cycle clock > 250 µs (for < V4.4 only factor 2, 4, 8 and 16 for all PN1 send cycle clocks)
[2] PN1: Onboard PROFINET IO interface for D435-2 DP/PN, D445-2 DP/PN and D455-2 DP/PN
[3] PN2: Optional second PROFINET IO interface with CBE30-2 (as of V4.3)

Figure 7-37 Cycle clock assignment, version 2

7.4.6 Synchronization of cycle clock and bus systems

Description

Synchronization between the slower and faster bus systems occurs automatically, whereby the cycle clock of Servo_fast is always an integer multiple of the servo.

Example of cycle clock and bus synchronization

This example shows how the individual cycle clocks behave during a cycle.

- Fast servo cycle: 250 µs
- Fast IPO cycle: 500 µs
- Slow servo cycle: 1 ms
- Slow IPO cycle: 2 ms

Cycle clock	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Time	0000	0250	0500	0750	1000	1250	1500	1750	2000	2250
Start faster, servo_fast	x	x	x	x	x	x	x	x	x	x
Start faster, IPO_fast	x		x		x		x		x	
Start slower, servo	x				x				x	
Start slower, IPO	x								x	

7.5 Time allocation in the round robin execution level

The time *remaining* after high-priority user and system tasks have been processed is used by the **MotionTasks** and the **BackgroundTask**.

This time is allocated to the BackgroundTask and the MotionTasks according to the *round robin principle*.

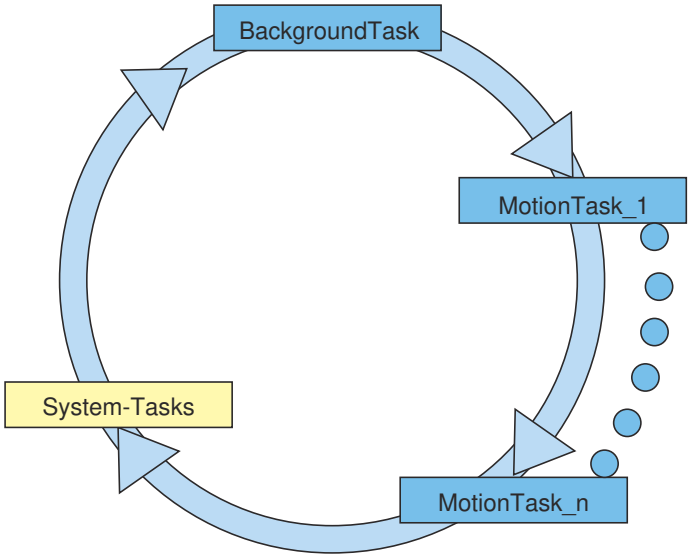


Figure 7-38 Overview of the time allocation in the round robin execution level

Round robin principle

Computing time that is not consumed by higher priority tasks is distributed over the remaining tasks in the round robin principle (Background Task, MotionTasks and Systemtasks); this means that these tasks run one after the other. Thus a quasi parallel (quasi "concurrent") processing of these tasks take place.

The next task is always started when the previous task gives up the computing time (task completed or in the "wait" state). The computing time of a round robin task is also limited to the maximum number of servo cycle clocks. In the next round robin cycle the computing time is continued at the point of interruption. The system ensures that a round robin task always gets at least the computing time of a servo cycle clock. BackgroundTasks can be favored so that the computing time of multiple successive servo cycle clocks can be made available to these tasks (slider execution system).

Additional tasks

In addition to the user program tasks (BackgroundTask, MotionTask), other system tasks (e.g. for communication) that require computing time in the round robin cycle also run in the round robin execution level.

Sequence

In general, the tasks run successively in the round robin cycle. (The sequence is *non-deterministic* and can, for example, change each time via a download, PowerON.)

Time allocation

A task can successively run a specified number of servo cycle clocks (remaining runtime) before it must transfer the computation time to the next task. However, it can also be transferred beforehand to the next task if it "does not have anything to do".

For the constructed case that all tasks in the round robin execution level have already been handled once, the first task is restarted. In this way, there is *no* IDLE time.

There are times in which no user program is being executed. This, for example, is the case when the BackgroundTask and the MotionTask are very brief (shorter than the remaining runtime in a servo cycle clock); the system tasks will then use the remaining runtime.

Restart / process image

After finishing, the BackgroundTask is not started again until the process image has been updated. The update is performed in the **ServoTask**.

Performance

A continuous loop in a MotionTask, without wait commands or synchronous motion commands, causes the MotionTask to use its maximum computation time. Consequently, the cycle of a BackgroundTask is additionally utilized (a larger part of the computation time goes into the MotionTask). Consequently, the system tasks in the round robin level are called at longer time intervals, which, for example, may influence the communication.

Note

MotionTasks with (continuous) loops without _waitTime (0 s) burden the round robin execution level as the MotionTask uses two complete servo cycle clocks.

During the allocation of the round robin execution level for the BackgroundTask and MotionTasks, the BackgroundTask in a complete round robin cycle receives at least one time slice, a MotionTask receives two time slices (if, for example, they are required because of a continuous loop).

Cyclic MotionTasks

Note when Motion Tasks should run cyclically:

If you want to use continuous loops in MotionTasks, then issue, for example, a **_waitTime(0s)** in each cycle. This MotionTask then passes to the following MotionTask.

Note

MotionTasks with (continuous) loops without _waitTime (0 s) burden the round robin execution level as the MotionTask needs two complete servo cycle clocks.

If you want a MotionTask to be executed cyclically, it is better to end the task in the normal way and restart the MotionTask again from a different task (e.g. BackgroundTask) rather than use a continuous loop or a step command at the start of the program. This also has advantages for a download during **RUN**.

See also

Optimizing the execution system (Page 600)

7.5.1 Setting of the time allocation

You can use a slider accessible via the **Time allocation** button for **MotionTasks** or for the **BackgroundTask** to set the chronological weighting of the BackgroundTask to the remaining tasks of the round robin execution level.

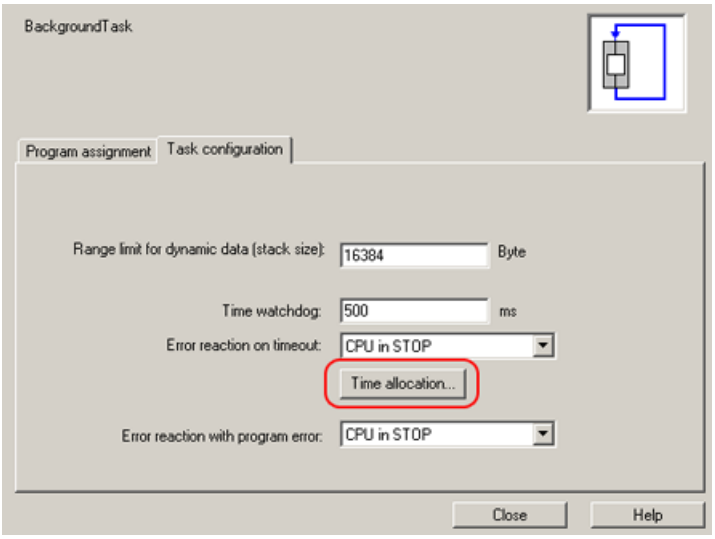


Figure 7-39 Task configuration in SCOUT

Specifying the time allocation in the round robin execution level

1. Select the **Task configuration** tab in the configuration window of a **MotionTask** or the **BackgroundTask**.
2. Click **Time allocation**.
   The Time Allocation in the Round Robin Execution Level window opens.
3. Use the slider to set the ratios of the computation times.
4. Click **OK** to confirm.

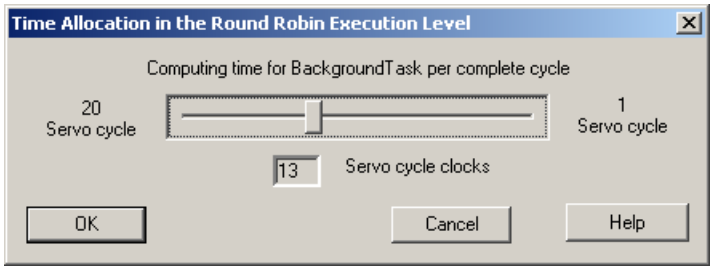


Figure 7-40 Time allocation in the round robin execution level

Time allocation in the round robin execution level

This is where you specify the ratio of the time allocation between the BackgroundTask and all other round robin tasks (MotionTasks and Systemtasks, see Execution levels/tasks (Page 199)).

The BackgroundTask gets a (maximum) of n-times the computing time of a different round robin task (all other round robin tasks, e.g. MotionTask_1, each have the same maximum share of computing time). You can set this factor with the slider.

When the BackgroundTask has run to an end, it will be restarted when the inputs are updated, at the earliest (after the next servo cycle).

If the computing time of the BackgroundTask is set high, and thus disadvantages the other round robin tasks then, for example, this will slow the communication to SCOUT/OP. Such a setting should only be made for extremely time-critical applications.

Table 7-4 Effect of computation time allocation to MotionTasks and BackgroundTask

Increased computation time for	**Effect**
MotionTasks	BackgroundTask takes longer to run from beginning to end; time monitoring may respond in extreme cases.
BackgroundTask	Programs in the MotionTask may take longer to execute under some circumstances.
The computation time for the BackgroundTask also includes the time required for acyclic communication (via PROFIBUS or PROFINET IO with IRT).	

7.5.2 Settings (examples)

To explain the time allocation in the round robin execution level, two examples are provided here for setting the slider.

Case 1: Slider at the far right-hand side

The BackgroundTask runs for at least one servo cycle clock.

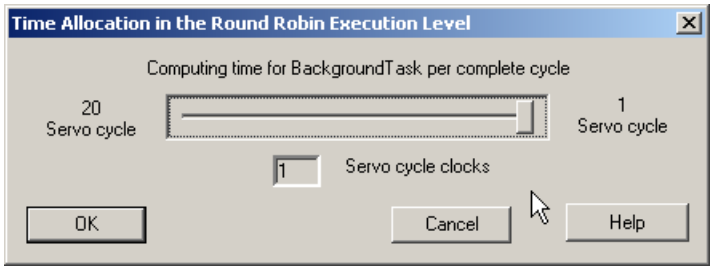


Figure 7-41 Time allocation in the round robin: Setting of the computation time for MotionTasks

In this setting, the BackgroundTask runs for one servo cycle clock, then all MotionTasks run (each MotionTask runs for maximum two servo cycle clocks, unless it enters the suspend mode beforehand), then the BackgroundTask runs again for one servo cycle clock, etc.

Table 7-5 Example:

Servo cycle clock 1	Servo cycle clock 2-3	Servo cycle clock 4-5	Servo cycle clock 6	Servo cycle clock 7-8
Background Task	MotionTask 1	MotionTask 2	Background Task	MotionTask 1

Sample program:

MotionTask 1 and 2 run concurrently with the BackgroundTask;
servo cycle clock = 3 ms

- A counter in a While loop is incremented in the BackgroundTask (green line).
- A counter in a While loop is incremented in MotionTask 1 (red line).
- A counter in a While loop is incremented in MotionTask 2 (blue line).

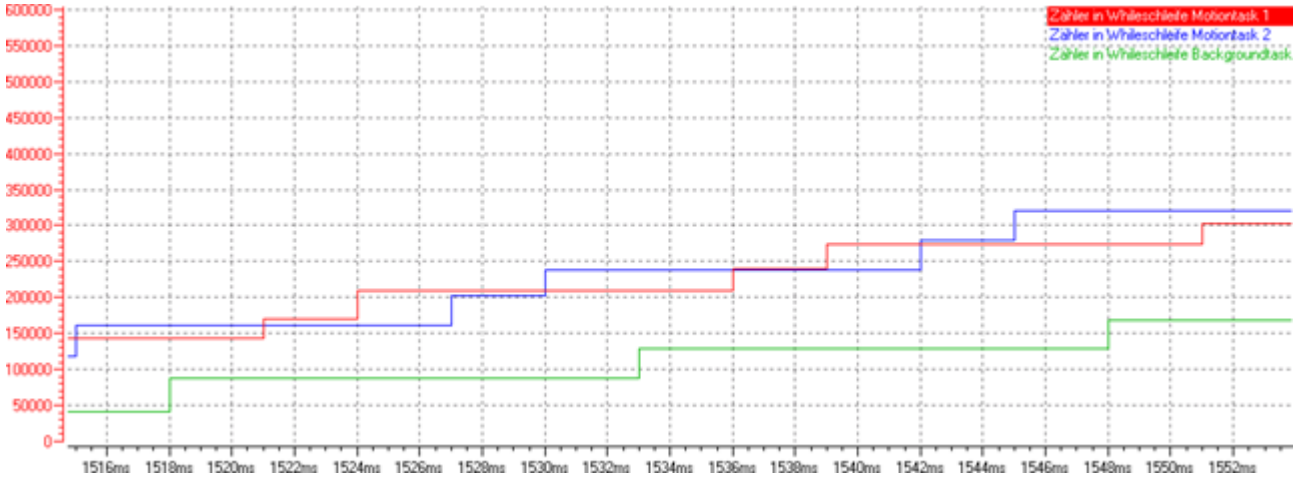


Figure 7-42 Execution example for the time allocation in the round robin: Setting of the computation time for MotionTasks

t = 1518 ms	BackgroundTask runs for one servo cycle clock
t = 1521 ms, 1524 ms	MotionTask 1 runs for two servo cycle clocks
t = 1527 ms, 1530 ms	MotionTask 2 runs for two servo cycle clocks
t = 1533 ms	BackgroundTask runs for one servo cycle clock

Case 2: Slider at the far left-hand side

The BackgroundTask runs for 20 servo cycle clocks.

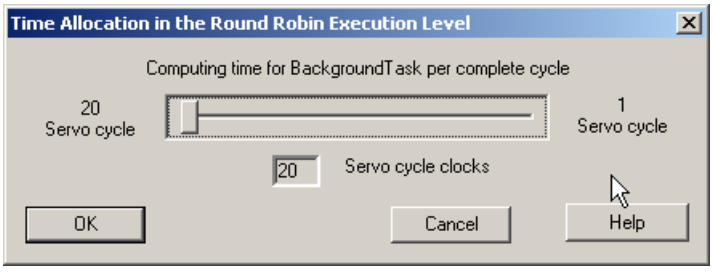


Figure 7-43 Time allocation in the round robin: Setting the computation time for BackgroundTask

In this setting, the BackgroundTask runs for 20 servo cycle clocks, then all MotionTasks run (each MotionTask runs for maximum two servo cycle clocks, unless it enters the suspend

mode beforehand), then the BackgroundTask runs again for another 20 servo cycle clocks, etc.

Table 7-6 Example:

Servo cycle clock 1-20	Servo cycle clock 21-22	Servo cycle clock 23-24	Servo cycle clock 25-40	Servo cycle clock 41-42
Background Task	MotionTask 1	MotionTask 2	Background Task	MotionTask 1

Sample program:

MotionTask 1 and 2 run concurrently with the BackgroundTask;
servo cycle clock = 3 ms

- A counter in a While loop is incremented in the BackgroundTask (green line).
- A counter in a While loop is incremented in MotionTask 1 (red line).
- A counter in a While loop is incremented in MotionTask 2 (blue line).

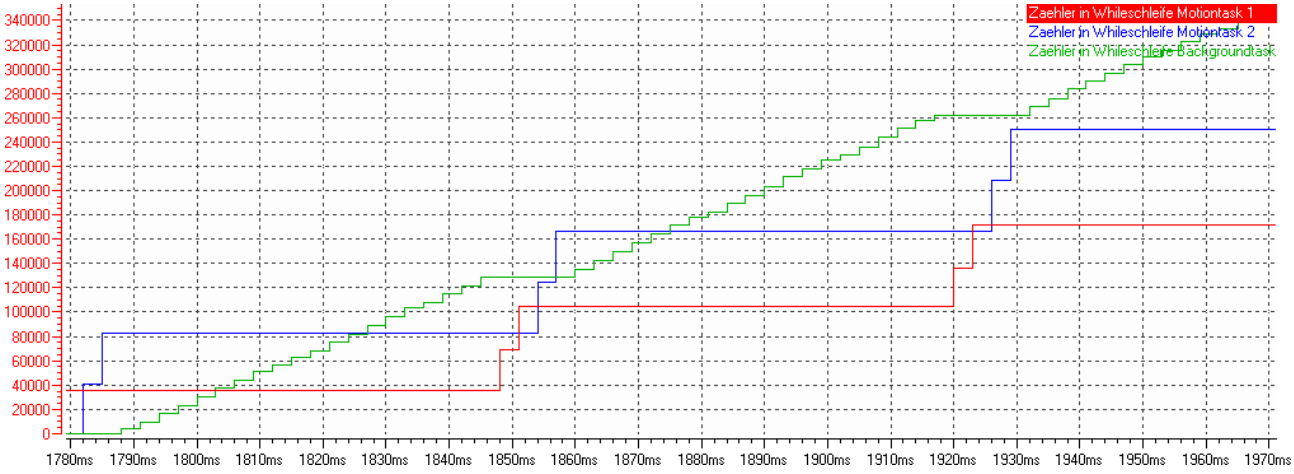


Figure 7-44 Execution example for the time allocation in the round robin: Setting the computation time for BackgroundTask

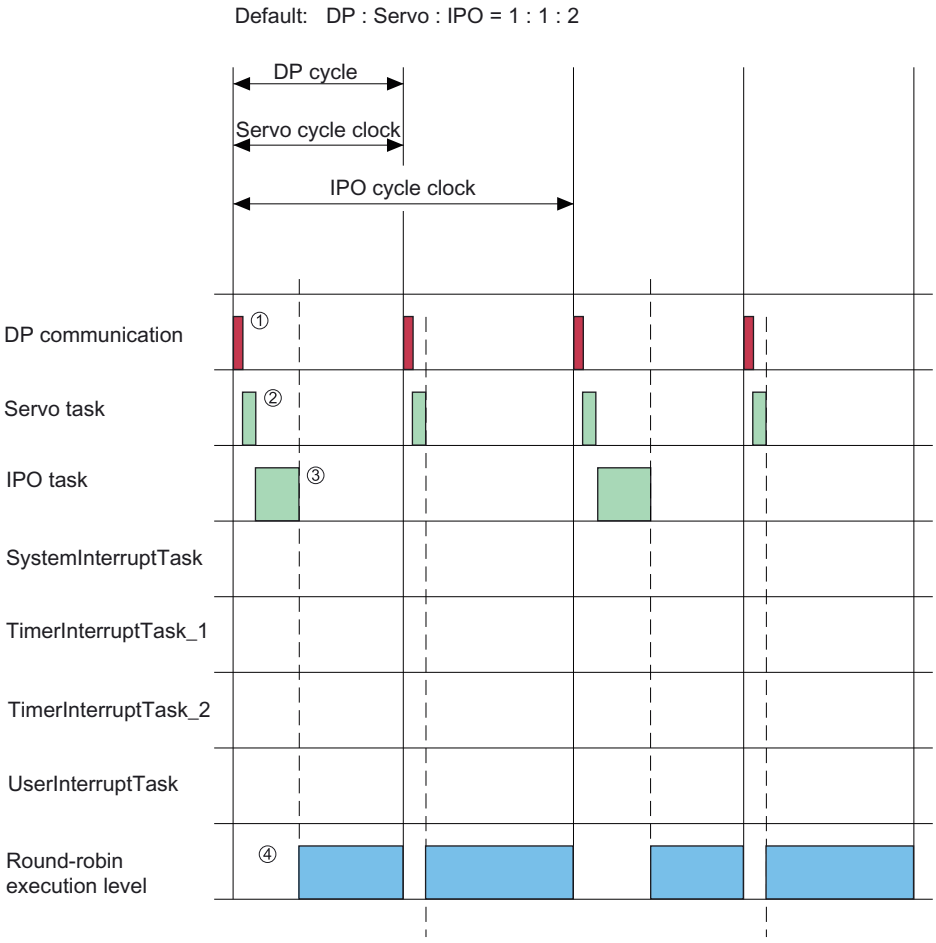
t = 1788-1845 ms	The BackgroundTask runs for 20 servo cycle clocks
t = 1848 ms, 1851 ms	MotionTask 1 runs for two servo cycle clocks
t = 1854 ms, 1857 ms	MotionTask 2 runs for two servo cycle clocks
t = 1860-1917 ms	The BackgroundTask runs for 20 servo cycle clocks

7.5.3 Processing of the tasks (examples)

The following examples illustrate the chronological execution of the various tasks.

The examples refer to PROFIBUS and apply to PROFINET IO with IRT analogously.

Example 1: Chronological execution when no InterruptTask is active



1 In the execution system, the cycle clocks have been selected as follows for this example:
DP cycle: Servo cycle clock: IPO cycle clock: 1:1:2
This means that the servo task is executed in each DP cycle and the IPO task is only executed in every second DP cycle.

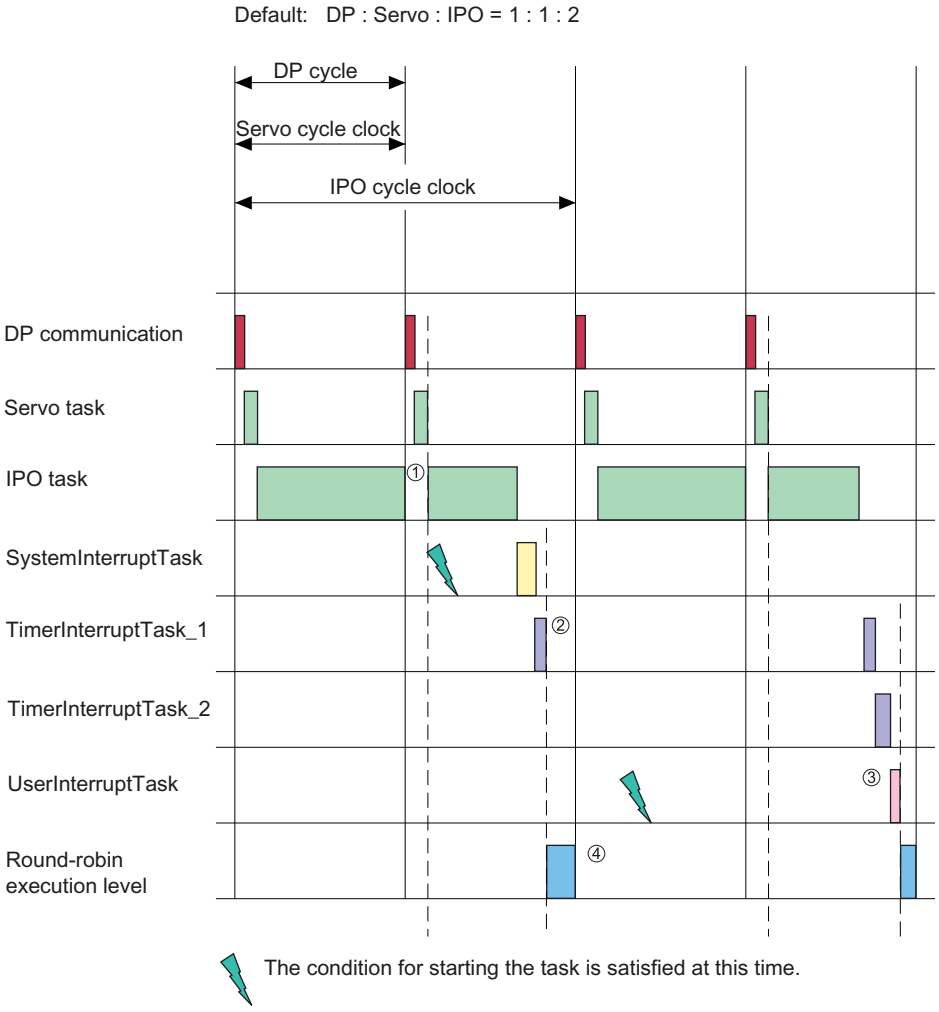
2 DP communication has the highest priority followed by the servo task.

3 IPO tasks are executed after servo tasks.

4 The round robin execution level is executed in the remaining time.

Figure 7-45 Chronological task execution, example: no InterruptTask active

Example 2: Chronological execution when an InterruptTask is active and IPOTask lasts longer than one servo cycle clock



1 The program executed in the IPO task lasts longer than the servo cycle clock. As a result, the IPO task is interrupted and the DP communication and the servo task are executed. Then, execution of the IPO task resumes.

2 After the IPO task is completed, the SystemInterruptTask is executed. Then the lower-priority TimerInterruptTask is started.

3 The UserInterruptTask is also not executed until the higher-priority tasks are completed, even if the condition for the UserInterruptTask was previously satisfied.

4 The round robin execution level is executed in the remaining time.

Figure 7-46 Chronological task execution, example: with InterruptTask active and IPOTask lasts longer than one servo cycle clock

Example 3: Special features for task change in multitasking

However the behavior of the multitasking with task changes, e.g., in case of wait commands or synchronous commands, must be taken into account when querying conditions in MotionTasks and also in the BackgroundTask.

Example:

Table 7-7 Example:

	BackgroundTask	**MotionTask1**	**MotionTask2**
Sources:	x=5	...	...
	... x=4 ...	if x <> 5 _enableAxis Endif	if x = 5 _stopAxis Endif
		...	...
	BackgroundTask	**MotionTask1**	**MotionTask2**
Execution sequence:	x=5		
	Task change →	if x <> 5 Condition not fulfilled endif	
		Task change →	if x = 5 Condition fulfilled Task change →
	x=4		
	Task change →	...	
		Task change →	...
			_stopAxis

Although the execution seems clear in MotionTask2, it may occur that the execution is other than intended due to a task change.

After the renewed change in the BackgroundTask (x=4), the if-queries have already been processed and the query results from before the task change are still valid. This means that although the condition x=5 is not fulfilled anymore at this time:

- _enableAxis is not executed in MotionTask1
- _stopAxis is executed in MotionTask2

The execution depends on when the task change is performed.

Note

To ensure multitasking runs smoothly, it is better not to use global variables from different tasks.

7.6 Task Trace overview

7.6.1 Using Task Trace

Description

The Task Trace is a tool for troubleshooting in the SIMOTION multitasking environment. The Task Trace offers the following options:

- Graphic display of the sequence of individual tasks and user events (generated using a program command).
- Trace of individual user tasks.
- Ability to configure the Task Trace using IT Diag or via the user program (system functions).
- Storage of the trace file on a memory card.
- Start of the SIMOTION Task Profiler as a separate application using IT Diag or the SCOUT device diagnostics.

For detailed information regarding the Task Trace, see the *Task Trace* function manual.

7.7 Isochronous I/O processing on fieldbus systems

The general sequence of isochronous I/O processing in a control system with I/O connection via a fieldbus system is presented below:

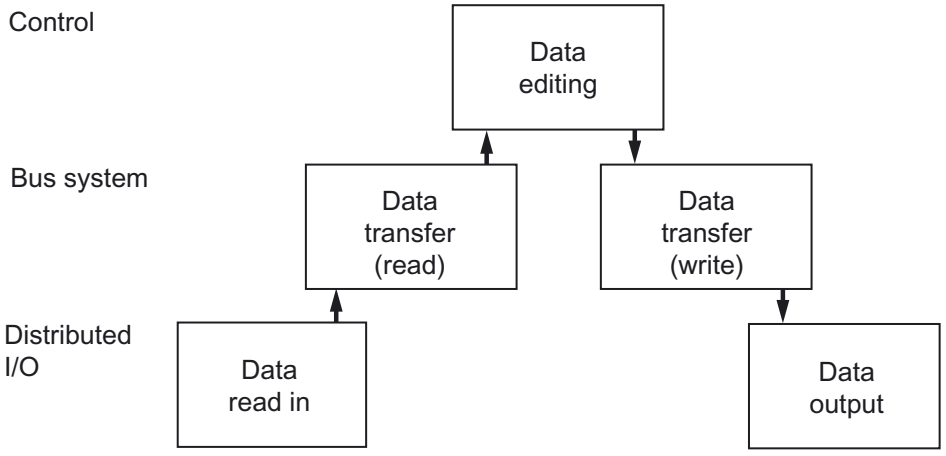


Figure 7-47 Data flow of isochronous I/O processing on fieldbus systems

In an isochronous system, the individual actions are chronologically synchronized with each other. Thus, short and reproducible response times (i.e. with the same length) are achieved.

Cycle clock and time reference are preset by an isochronous, i.e. clocked bus system.

7.7.1 Data protocol on PROFIBUS DP

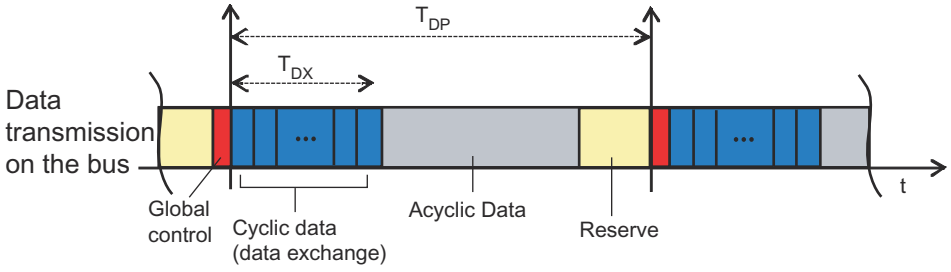


Figure 7-48 Data protocol on PROFIBUS DP

The data protocol on the bus contains the following:

- An isochronous **global control message frame (GC)** that defines the bus cycle clock
- **Cyclic data**: Data transmitted between the nodes in each cycle clock
  Cyclic data communication allows for especially short and reproducible process response times. The transfer of information is done *in each cycle*.
  A user program accesses this data via the process image or via direct access to the I/O.
- **Acyclic data**: Data transmitted only if required and in larger quantities
  The acyclic transmission is suitable especially for the non-time-critical transmission of larger data quantities; the data can be distributed to several tasks.
  Example of acyclic data: Alarms, diagnostic services and data sets
  A user program accesses this acyclic data, e.g. using the commands _readrecord, _writerecord.
- **Reserve**: Remaining time until the next global control.
  The residual time differs depending on the current running acyclic communication, and is a maximum of T_{DP}-T_{DX}.

7.7.2 Data protocol on PROFINET IO

Description

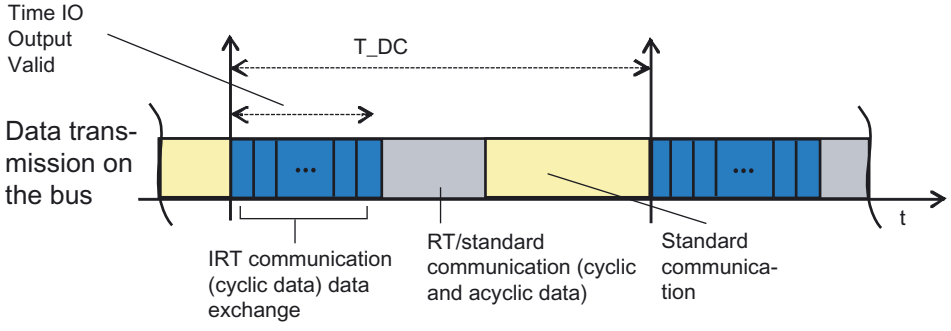


Figure 7-49 Data protocol on PROFINET IO

The data protocol on the bus contains the following:

- The IRT communication that transfers cyclic data in a planned time interval.
- The RT and standard communication in which the following data can be transferred:
  - Cyclic RTC message frames with which I/O data can be transferred, for example
  - Acyclic RTA telegrams with which alarm data can be transferred, for example
  - NRT data (standard communication), standard Ethernet telegrams with which all standard Ethernet-based protocols can be transferred
- Standard communication (last interval prior to synchronization telegram); here only communication can take place that is concluded by the time the synchronization telegram is used.

TIME IO Output Valid describes the time during which the new output data are available.

7.7.3 Isochronous data processing

Isochronous application

With an isochronous application, the processing cycle clock of the controller and any drives connected or distributed I/O are synchronized to the bus cycle.

Isochronous data transmission on PROFIBUS DP

For PROFIBUS the bus cycle clock is synchronized to the Global Control Telegram (GC). The GC embodies the bus cycle clock and is also the system cycle clock for the entire system.

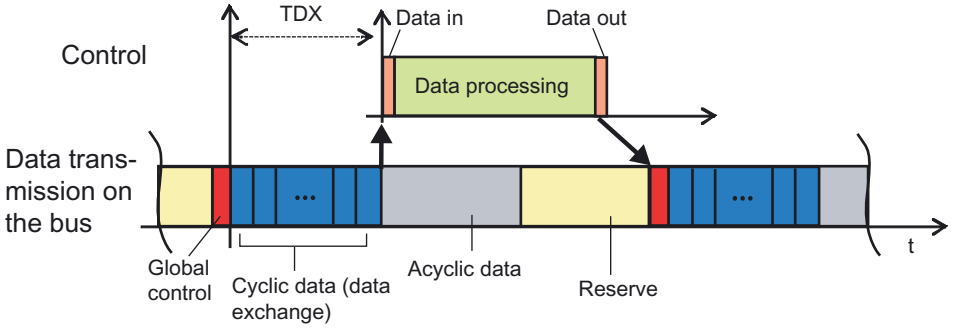


Figure 7-50 Structure of a bus cycle for PROFIBUS DP

Isochronous data transmission on PROFINET IO

For PROFINET IO the processing cycle clock of the controller is synchronized to the IRT communication.

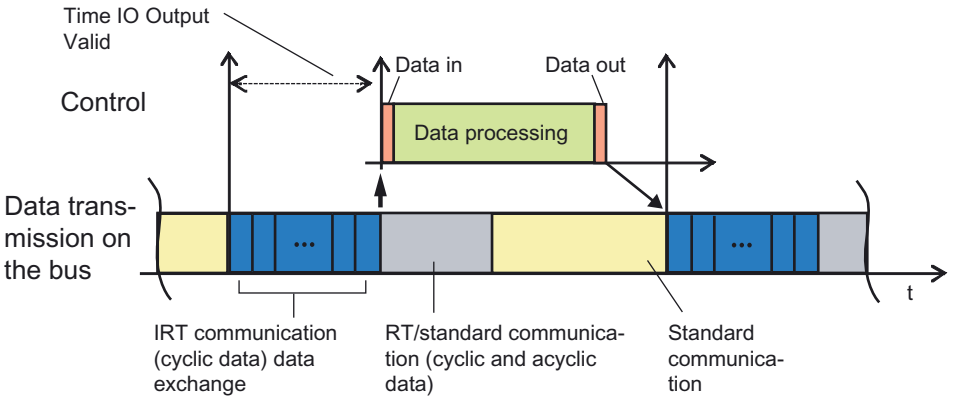


Figure 7-51 Structure of a bus cycle for PROFINET IO

The data transmitted via cyclic data communication can thus be processed isochronously.

Isochronous data acquisition with PROFIBUS DP

In an isochronous complete system, data acquisition, data processing and data output refer to the total system cycle clock.

The application is started when the cyclic data transfer has been completed (T_{DX}).

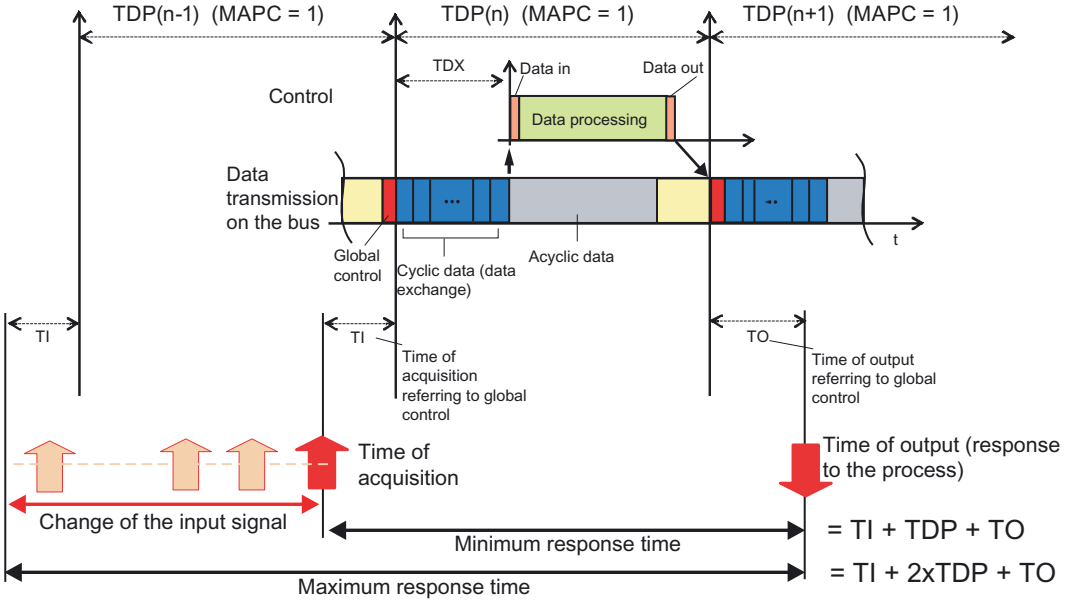


Figure 7-52 Calculation of the reaction times (PROFIBUS DP)

Isochronous data transmission with PROFINET IO

The application is started when the cyclic data transfer has been completed (Time IO Output Valid).

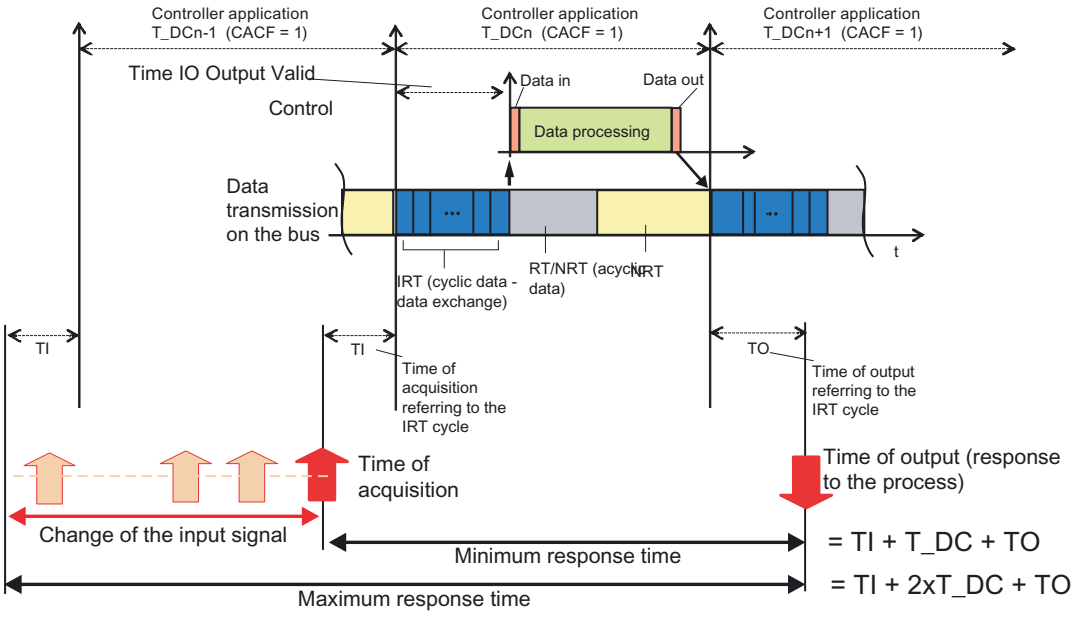


Figure 7-53 Calculation of the reaction times (PROFINET IO)

Calculation of the reaction times (PROFIBUS DP and PROFINET IO)

The read-in process in the distributed I/O must be brought forward, offset by time T_I, so that a consistent state of the inputs can be read in at the starting time of the new bus cycle.

The time T_I comprises at least the signal processing, and conversion time on the electronic modules, and in the case of a modular ET 200 I/O system also the transmission time of the inputs on the ET 200 backplane bus. In general T_I is selected in such a manner that it is the same for all modules, the slowest module determines the time in this regard.

The time T_O ensures that the process reactions of the user program (data processing) are switched through simultaneously and consistently to the "terminals" of the distributed I/O on the bus. analogously for drives the set point is made available for the drive control. The time T_O comprises at least the time for the cyclic data exchange of all nodes on the bus; in the case of a modular ET 200 I/O system the transmission time of the outputs on the ET 200 backplane bus and the signal processing, and conversion time on the electronic modules.

Reaction times or dead times (for drives)

By synchronizing the individual cycles it is possible to read the input data in the cycle clock "n-1", to transmit and process the data in the cycle clock "n", and to transmit the calculated output data at the beginning of the cycle clock "n+1", and to connect the output data to the "terminals" in the same cycle clock.

PROFIBUS

This results in a real process reaction time of:

- $T_I + T_{DP} + T_O$ minimum
- $T_I + 2 \cdot T_{DP} + T_O$ maximum

PROFINET

This results in a real process reaction time of:

- minimum $T_I + T_DC + T_O$
- maximum $T_I + 2 \cdot T_DC + T_O$

7.7.4 Dynamic response with respect to data processing in the control

Data processing in the ServoSynchronousTask

- For data processing in the ServoSynchronousTask, for the bus cycle clock: Servo = 1 : 1 the setting can produce a processing time of one bus cycle clock in the controller.
  This applies, for example, for the setting of output data, e.g. DO or AO, depending on the input signals.

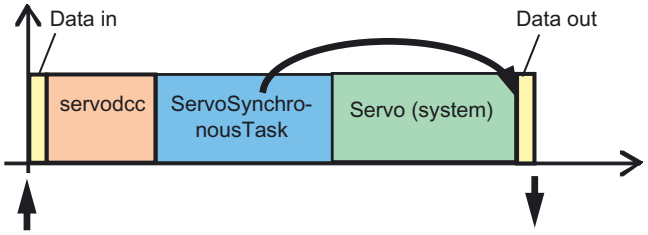


Figure 7-54 Data processing in the control - setting the output data

- When influencing axis data/axis motions, the processing time depends on the fact if the data is already active in the servo (e.g. superimposed setpoint or output value).

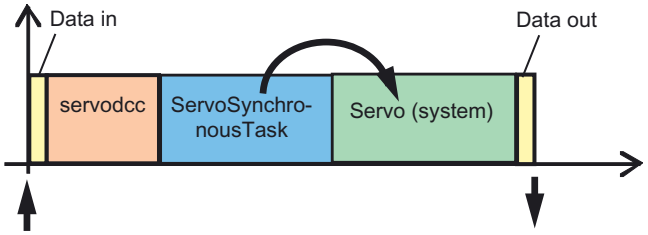


Figure 7-55 Data processing in the control - influencing motion data

- If the data or commands become effective not until the IPO (e.g. issuing of motion commands), the output data on the outputs only become effective with the next servo.
  Note: To keep the runtime of the ServoSynchronousTask low (and thus the time until Data Out), it is preferable to implement the programming of such tasks in the IPOSynchronousTask.

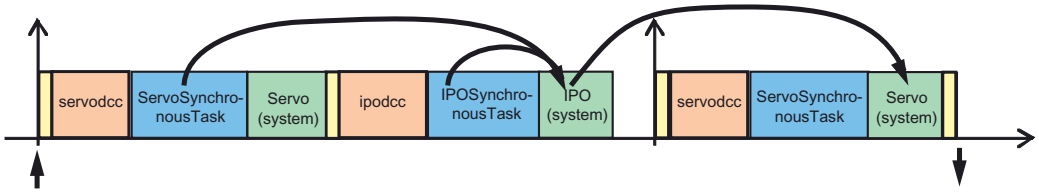


Figure 7-56 Data processing in the control - output data effective in the IPO

Data processing in the IPOSynchronousTask

- In the data processing in the IPOSynchronousTask, the output data on the I/O become effective with the next servo.

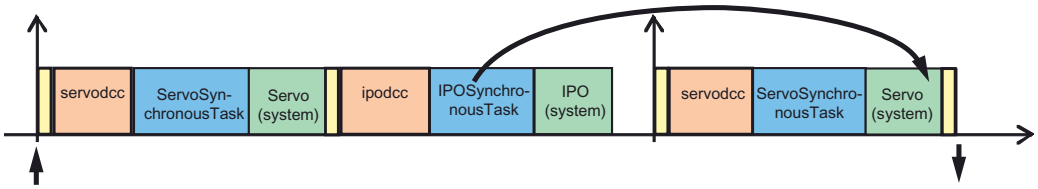


Figure 7-57 Data processing in the control - output data effective in the next servo

- Or: Motion data in the next IPO or servo (see above)

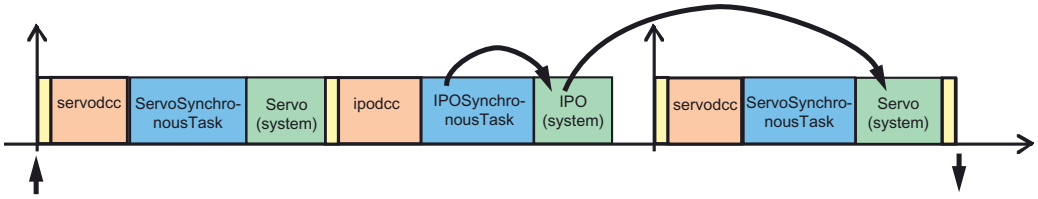


Figure 7-58 Data processing in the control - output data effective in the IPO - influencing motion data

In those cases where the output data of the I/O are only transmitted in the next servo, the reaction time is increased by one bus cycle clock each (for bus cycle clock: servo : IPO = 1 : 1 : 1).

Scenario	Recommendation for user program task
Fast terminal-terminal response on I/O	Use ServosynchronousTask
Fast influencing of the setpoint (servo level)	Use ServosynchronousTask
Switching/superimpositioning the motion (e.g. _stop, _move, _pos, ...) e.g. print mark	Use IPOsynchronousTask

As of V4.1 in exceptional cases the IPO share can also be calculated in the servo, in this regard see **Motion control / interpolator** in the **TO axis** manual.

7.7.5 Dynamic response with respect to data transmission from the acquisition to the processing

When using a PROFIBUS or PROFINET bus system, the specified times for data transmission (T_{DX}) must be taken into account in the calculation of the system cycle clock. Small T_{DX} allow for shorter cycle clocks or more processing time for the application with the same cycle clock length.

- For PROFIBUS DP T_{DX} is shown by the HW config.
- On SIMOTION D, a transmission time in the range of 125 µs ≦ T_{DX} ≦ 375 µs is set automatically on PROFIBUS integrated depending on the degree of extension of the SINAMICS project (I/O extension on SIMOTION D and SIMOTION CX32).
- For PROFINET IO $T_{DX} = 0.5 \bullet T_{DP}$ is set automatically

7.7.6 Dynamic response for data acquisition and data output

When configuring time T_O take T_{DX} into consideration. The following always applies: $T_{DX} \leqq T_O$.

- The times can be set in HW Config.
- When using TM15/TM17 High Feature modules for acquisition and output, the following times are to be set:
  - For the acquisition: T_I + 1 DRIVE-CLiQ cycle (typically 125 µs)
  - For the output T_O + 1 DRIVE-CLiQ cycle (typically 125 µs)

See also

Determination of Tdp, Ti and To using HW Config for ET 200 I/O devices on the PROFIBUS (Page 291)

7.7.7 Determination of Tdp, Ti and To using HW Config for ET 200 I/O devices on the PROFIBUS

In HW Config in the **Isochronous mode** screen form (called from the **Edit > Isochronous mode** menu), the times for T_{DP}, T_I and T_O can be determined.

The dialog box gives an overview of the parameters and times set for the isochronous mode of the involved objects PROFIBUS, slaves and modules in the respective slaves.

All times in the dialog are specified in milliseconds.

- T_I/T_O setting
  - **in the network** means that the times T_I and T_O are centrally set "same for all slaves"
  - **in the slave** means that the times T_I and T_O are set individually on each slave
- T_I, T_O, T_{DP}
  Shows the currently set or calculated values T_I, T_O and T_{DP} for the selected DP master system.

For a detailed description of the screen form and all other parameters, please refer to the online help for HW Config.

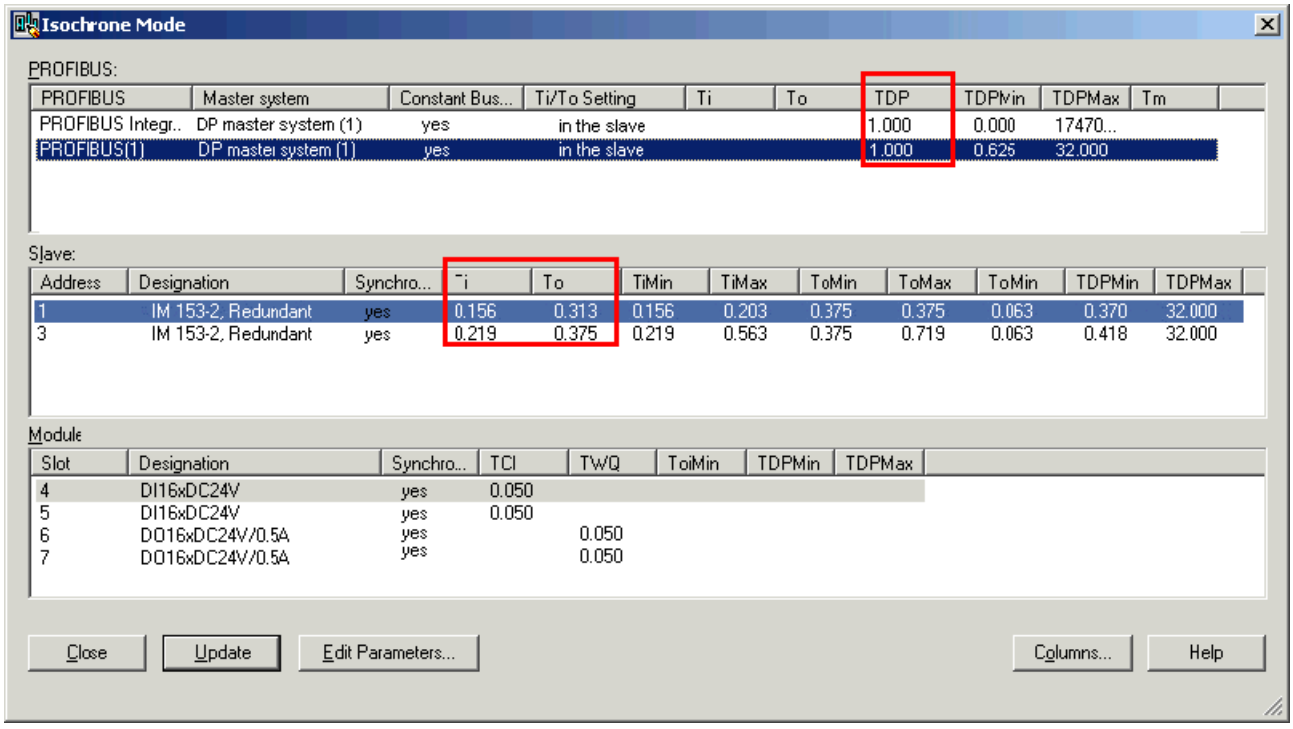


Figure 7-59 Isochronous mode screen form in HW Config

Note

Full "terminal-to-terminal" support of isochronous mode is only possible if all components within the sequence support the "isochronous mode" system property. When selecting from the PM10 catalog or the hardware catalog of HW Config, pay attention to the entry **Isochronous mode** in the information field of the module.

A current list of isochronous I/O modules can be found on the Internet under http://support.automation.siemens.com/WW/view/de/14747353.

7.7.8 Handling cycle clock scaling

The following must be observed for cycle clock scaling:

- At the end of an IPO synchronous task the process image is output with the next possible servo (Data Out) (= reaction time optimized). When scaling the servo cycle clock to the IPO cycle clock sooner or later this can cause the data of one or more servo cycle clocks to be output within an IPO cycle clock, if the I/O access are executed via the IPOSynchronousTask.
- At the end of the servo execution level, the process image of the ServoSynchronousTask is output with the next possible bus cycle clock (= reaction time optimized).
  - For PROFINET and downscaling PROFINET cycle clock to servo cycle clock this can cause the data of one or more bus cycle clocks to sooner or later be output within one servo cycle clock if the I/O accesses are executed via the ServoSynchronousTask and the runtime of the server execution level fluctuates beyond one bus cycle clock.
  - For PROFIBUS the data are always output with the first bus cycle clock as the servo execution level must always be concluded with the first bus cycle clock.

Thus with a different runtime of the servo execution level in the individual cycle clocks, the terminal-terminal time may vary.

If an always constant reaction time is to be achieved instead of a reaction time optimized behavior, the following must be set:

- For PROFIBUS:
  - A reduction ratio servo: IPO = 1 : 1 so that the I/O accesses from the IPOSynchronousTask are always implemented in isochronous mode.
    Comment: I/O accesses from the ServoSynchronousTask are always isochronous for PROFIBUS
- For PROFINET:
  - A reduction ratio bus cycle clock : Servo: IPO = 1 : 1 : 1 so that the I/O accesses from the IPOSynchronousTask are always implemented in isochronous mode
  - A reduction ratio bus cycle clock : servo = 1 : 1 so that the I/O accesses from the ServoSynchronousTask are always implemented in isochronous mode

7.8 Integrating DCC into the SIMOTION execution system

7.8.1 Introduction

Introduction

Drive Control Chart (DCC) allows you to implement your regulating and control tasks in a drive-related manner. For this there is a set of blocks (DCB) available to you that you can interconnect graphically via a configuration tool (DCC editor) in so-called charts. The DCBs are available to you in the form of a library (DCBLIB).

Note

One exception in the DCC tasks does not call up a SIMOTION fault task.

Thus the charts created can be executed on the SIMOTION platforms as well as on SINAMICS drives.

Only DCC for Simotion is described below.

Additional references

DCB lib DCC editor description

7.8.2 Sequence model for DCC blocks (DCB)

Description

The individual DCC blocks (DCB – Drive Control Block) are organized as runtime groups in the DCC editor. You can freely assign these runtime groups to the DCC tasks in the DCC editor (task T1 to T5). The DCC tasks in the DCC editor correspond to the DCC tasks in the SIMOTION execution system.

Task in the DCC editor	Execution level	Task
T1	Servo	servoDcc
D2	IPO	ipoDcc
T3	IPO_2	IpoDcc_2
T4	DccAux	dccAux
T5	DccAux_2	dccAux_2

Execution groups can be activated or deactivated via binary outputs of blocks. Detailed description in the editor.

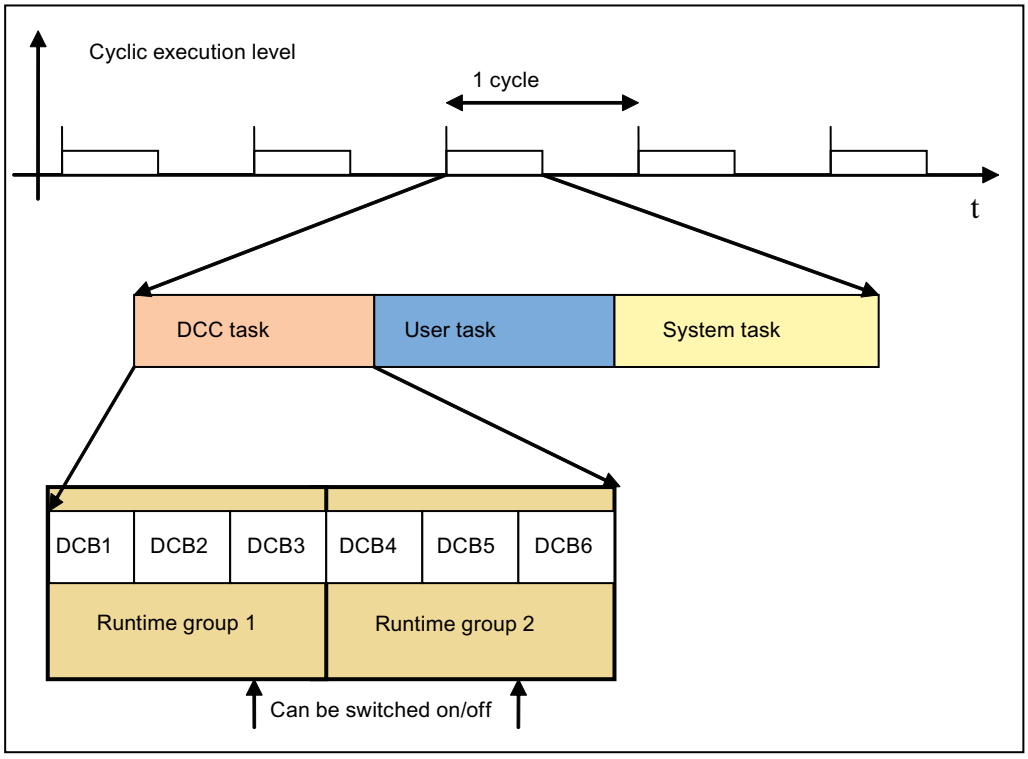


Figure 7-60 Sequence model of the runtime groups for DCBs

In the user model, the DCC task assigns itself in the execution level next to the user program tasks (execution environment for user programs in ST, MCC, LAD/FBD) and the system tasks. If required, the DCC task is created by the Engineeringsystem (ES); this means it is *not* present in the system as standard.

Execution of the DCC diagrams

The execution of DCC diagrams is linked to the task status but rather to the status of the runtime group assigned to it. They are enabled and disabled depending on the mode changes.

Mode	Action
STARTUP->RUN	ON runtime groups
SHUTDOWN->STOPU	OFF runtime groups
SHUTDOWN->STOP	OFF runtime groups

All other changes in the mode cause no changes to the runtime groups.

See also

servoDccTask in the servo level (Page 296)

ipoDcc Task in the IPO level (Page 297)

ipoDcc_2 Task in the IPO2 level (Page 298)

Execution levels for DccAux and DccAux_2 (Page 299)

7.8.3 servoDccTask in the servo level

Description

The T1(DCC) task runs in the servo level of the SIMOTION execution system. The servo level is called synchronous to the data exchange with the isochronous I/O and used primarily for fast control tasks.

- At the start, the cyclic I/O data for the technology objects and servo-synchronous I/O variables is copied from the interface to the isochronous I/O in the execution system.
- At conclusion, the cyclical I/O data is copied again from the execution system and, for example, transferred with PROFIBUS to a PROFIBUS node.

The following graphic shows the chronological sequence of a task in the servo level.

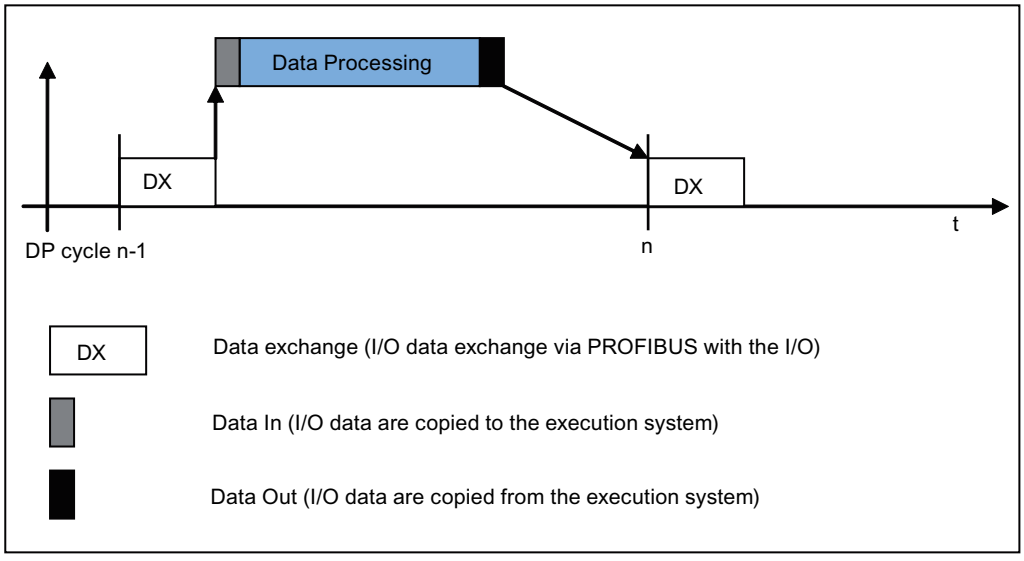


Figure 7-61 T1(DCC) task in the servo level

The T1(DCC) task is automatically started and executed at the begin of the servo level.

The following graphic shows the sequence of the tasks in the servo level.

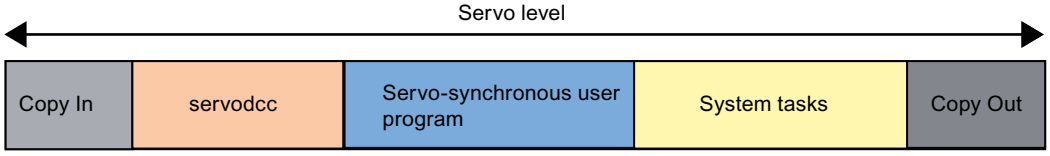


Figure 7-62 Servo level with T1(DCC) task

Level overflow in the servo level

Tasks in the servo level run with the same priority and so do not interrupt each other. If all tasks of the servo level cannot be calculated in a single cycle, a level overflow occurs and the system enters STOP mode and the start-up lock is set. Only after a new startup (power on/off) or download can the system return to RUN mode.

See also

Defining system cycle clocks (Page 250)

7.8.4 ipoDcc Task in the IPO level

Description

The IPO level runs with the same cycle time as the servo level or in an integer reduction ratio. Tasks in the servo level have a higher priority and so can interrupt tasks in the IPO level. In addition, in the IPO level all I/O data not copied in the servo level are copied into the execution system. The data can come both from the isochronous and the non-isochronous I/O.

The command processing and the setpoint generation of the technology objects run in the system task of the IPO level.

The following graphic shows the chronological sequence in the IPO level.

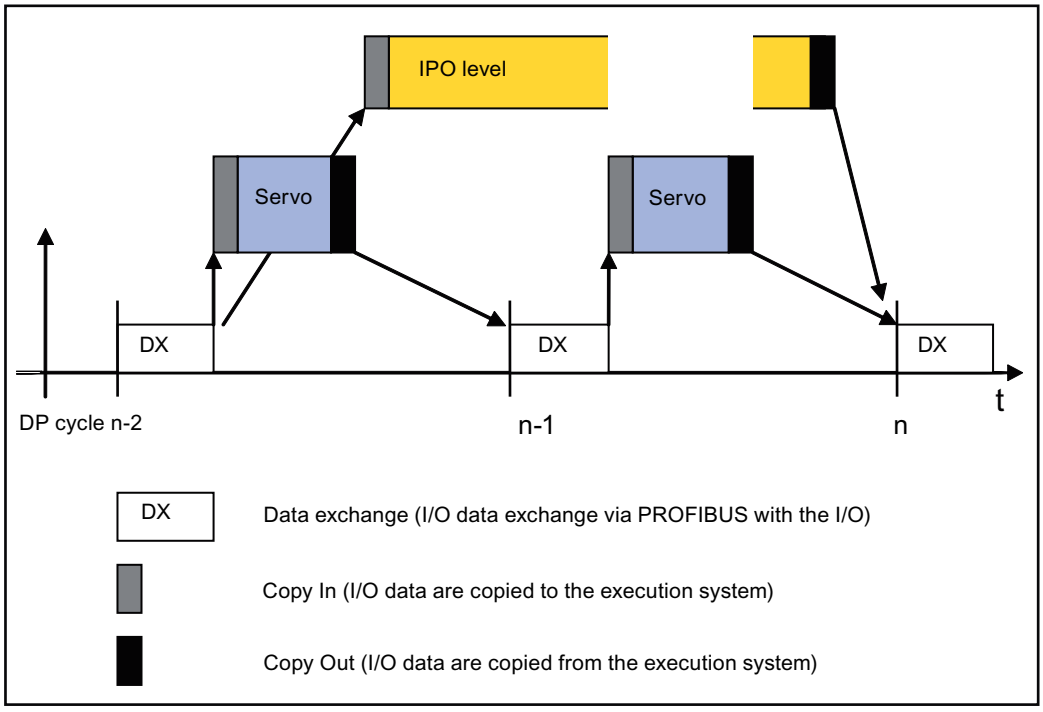


Figure 7-63 IPO level with T2(DCC) task

The T2(DCC) task is automatically started and executed at the begin of the IPO level.

The following graphic shows the execution sequence in the IPO level.

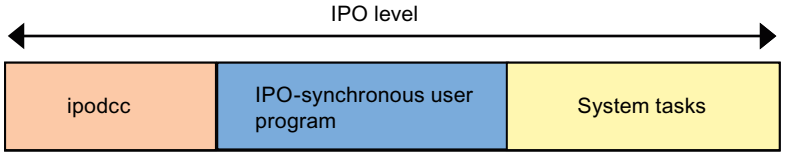


Figure 7-64 Execution sequence in the IPO level

Execution overflow in the IPO level

Tasks in the IPO level run with the same priority and so do not interrupt each other. A task in the higher priority servo level can at anytime interrupt the processing in the IPO level; this can possibly cause a level overflow, because not all tasks in the level can be processed.

With the standard behavior, for a level overflow the device enters into STOP mode and the start-up lock is set. Only after a new startup (power on/off) or download can the system return to RUN mode.

However, you can also configure the system for toleration of up to five successive overflows. You must set the behavior appropriately in the task configuration.

See also

Defining system cycle clocks (Page 250)

Timeouts and level overflows (Page 260)

SynchronousTasks (Page 224)

Isochronous data processing (Page 285)

7.8.5 ipoDcc_2 Task in the IPO2 level

Description

The cycle time of the IPO2 level is an integer multiple of the cycle time of the IPO level. However, it is not possible to set the IPO2 cycle clock equal to the IPO cycle clock.

The cycle time of the IPO2 level is shorter than that of the IPO level. Consequently, both the IPO level and the servo level can interrupt the processing.

The T3(DCC) task is assigned in the IPO2 level as follows:

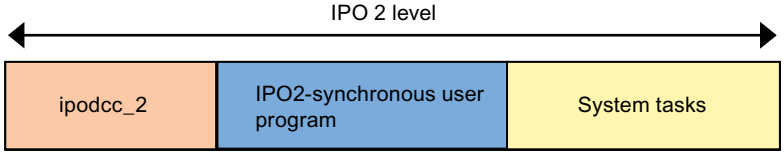


Figure 7-65 Order of processing in the IPO2 level

Level overflow in the IPO2 level

In the case of a level overflow, the IPO2 level behaves like the IPO level.

See also

ipoDcc Task in the IPO level (Page 297)

Defining system cycle clocks (Page 250)

7.8.6 Execution levels for DccAux and DccAux_2

Description

The two DCC execution levels, DccAux and DCCAux_2, are called synchronous to the system.

The DCCAux and DCCAux_2 levels have the following properties:

- DccAux has a multiple of the Ipo_2 cycle clock.
- DccAux_2 has a multiple of the DccAux cycle clock.
- DccAux has a higher priority than DccAux_2
- Synchronous trace recordings are possible.
- Up to five level overflows are tolerated (the default is 1). The current number of level overflows can be fetched.
- In the case of a level overflow, the DCCTask is calculated to completion in the next task.

Table 7-8 System variables for fetching level overflows

System variable	Data type	Meaning
_device.numberOfSummarized-TaskOverflow.DccAux	UDINT	Number of overflows of the DccAux level since system startup
_device.numberOfSummarized-TaskOverflow.DccAux_2	UDINT	Number of overflows of the DccAux_2 level since system startup

See also

Defining system cycle clocks (Page 250)

7.8.7 Data exchange between blocks

7.8.7.1 Data exchange between blocks (overview)

Overview

In the DCC editor, you can interconnect the connections of various blocks. Three basic scenarios must be differentiated for the data exchange between the blocks:

- You have connected connections of blocks with each other that lie in the same execution level.
- You have interconnected the output of one block with the input of a block in a higher execution level.
- You have interconnected the output of one block with the input of a block in a lower execution level.

See also

Data exchange between blocks in the same level (Page 300)

Data for blocks from a lower-priority level (Page 302)

Data for blocks from a higher-priority level (Page 305)

7.8.7.2 Data exchange between blocks in the same level

Description

If you have interconnect the blocks in the same execution level, the data will be transferred in accordance with the execution sequence of the blocks. The guarantees that the value is current at the input of the execution sequences.

If you have defined the execution sequence in accordance with the data flow, no additional dead time results in the level.

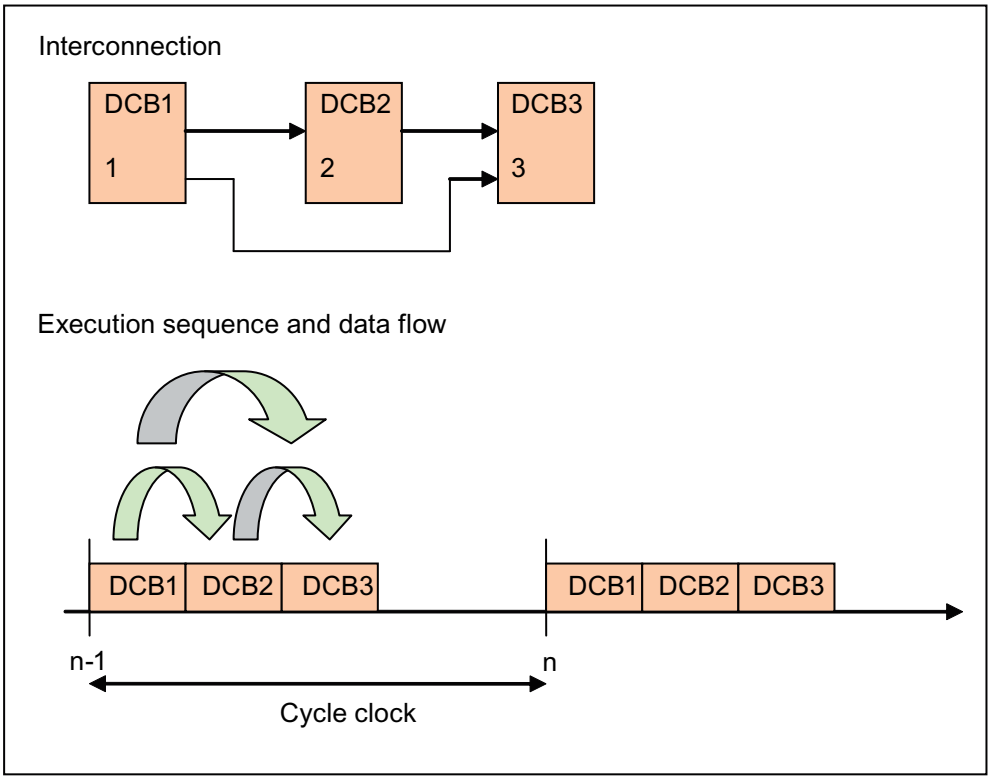


Figure 7-66 Interconnection sequence and execution sequence are identical

If the execution sequence of the blocks in the level does not correspond to the data flow, additional dead times result, because at the input access has been made to values from the previous cycle clock.

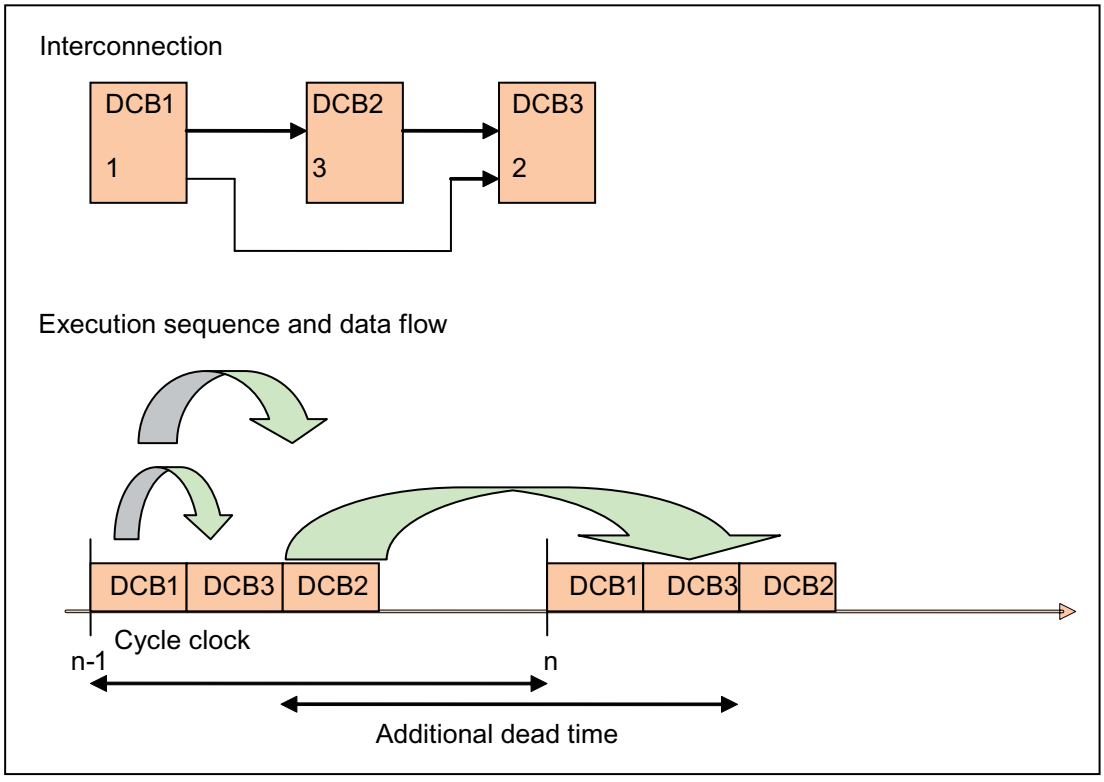


Figure 7-67 Interconnection sequence and execution sequence differ

To optimize the dead times, always orient the execution sequence on the data flow.

7.8.7.3 Data for blocks from a lower-priority level

Description

All blocks in an execution level must be calculated before their output values are made available to blocks at the input in a different execution level. Output values from a higher-priority level are accessed in a dedicated cycle clock (acceptance cycle clock) in order to ensure isochronism of input values. The following graphic illustrates the isochronous value transfer in a higher-priority level.

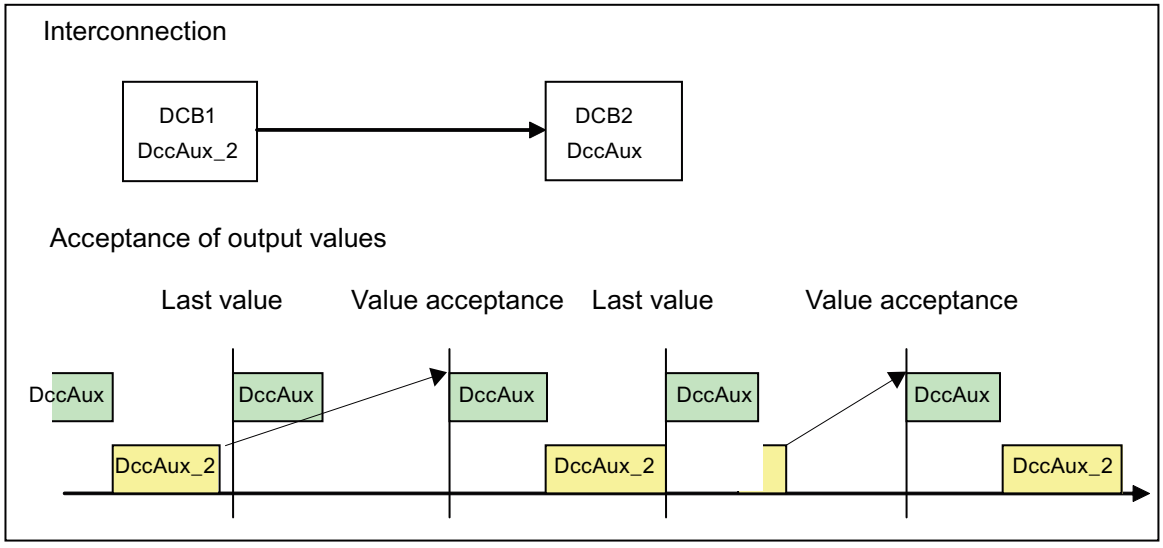


Figure 7-68 Example for the data exchange from a lower priority level

The cycle clock ratio between the levels is 1:2. The task in the higher-priority level always accepts the value in a second cycle clock, independent of whether the lower-priority task was already calculated to the end in the previous cycle clock.

Level overflow of the lower-priority level

If a level overflow of the lower-priority level occurs, the current values have not yet been calculated at the acceptance cycle clock. In this case, the old value is used for calculations at the input until the next acceptance cycle clock.

The following graphic illustrates the overflow behavior for a cycle clock ratio of 1:2. A level overflow for T2 occurs. This means the old values are transferred to the higher-priority task at the acceptance time (cycle clock 1). A new value is accepted at the next acceptance time, namely two cycle clocks later.

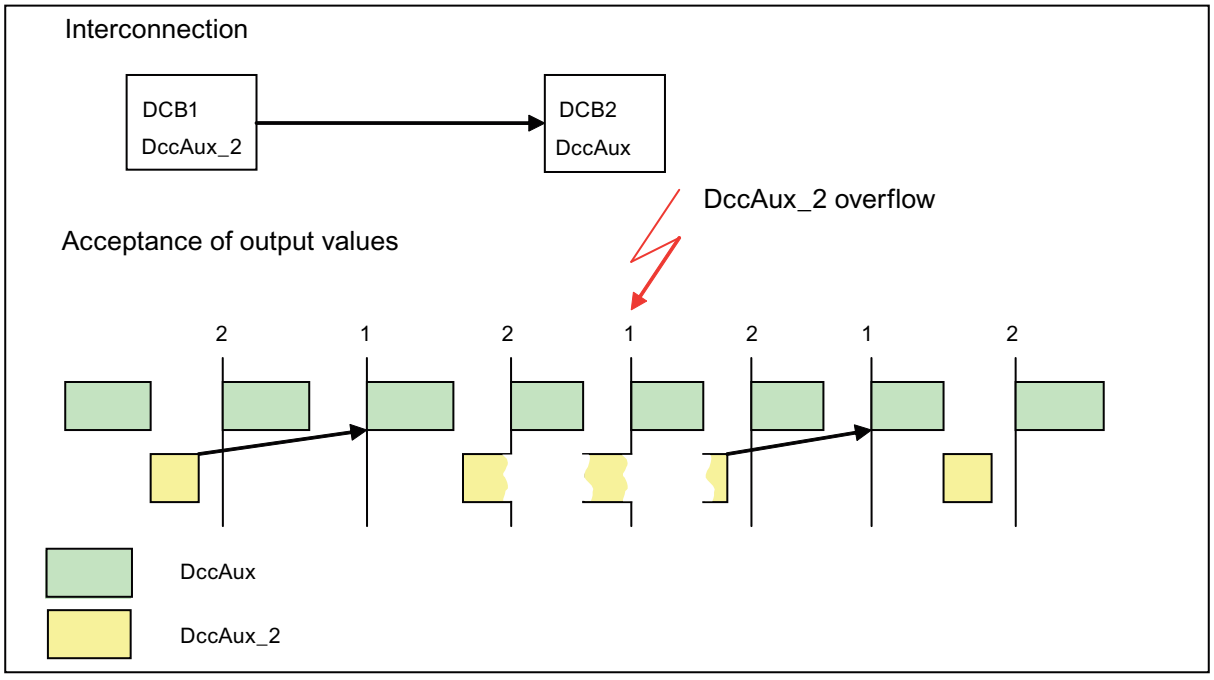


Figure 7-69 Data exchange from a lower-priority level with overflow

The described overflow scenario is true only for an ideal system. In a real application, user tasks and system tasks also run in a level together with the DCC task.

Example consideration of user programs and system tasks

If overflow occurs during the processing of the iposynchronous UP or of the IPO system task, the values have already been calculated in the IPoDCC task and can be transferred to the higher-priority level. However, no new value is calculated in the next cycle clock. If the ipoDCC task already causes an overflow, a correct value will only be accepted again two cycle clocks later. In the two cycle clocks between, the value is accepted delayed by one cycle clock.

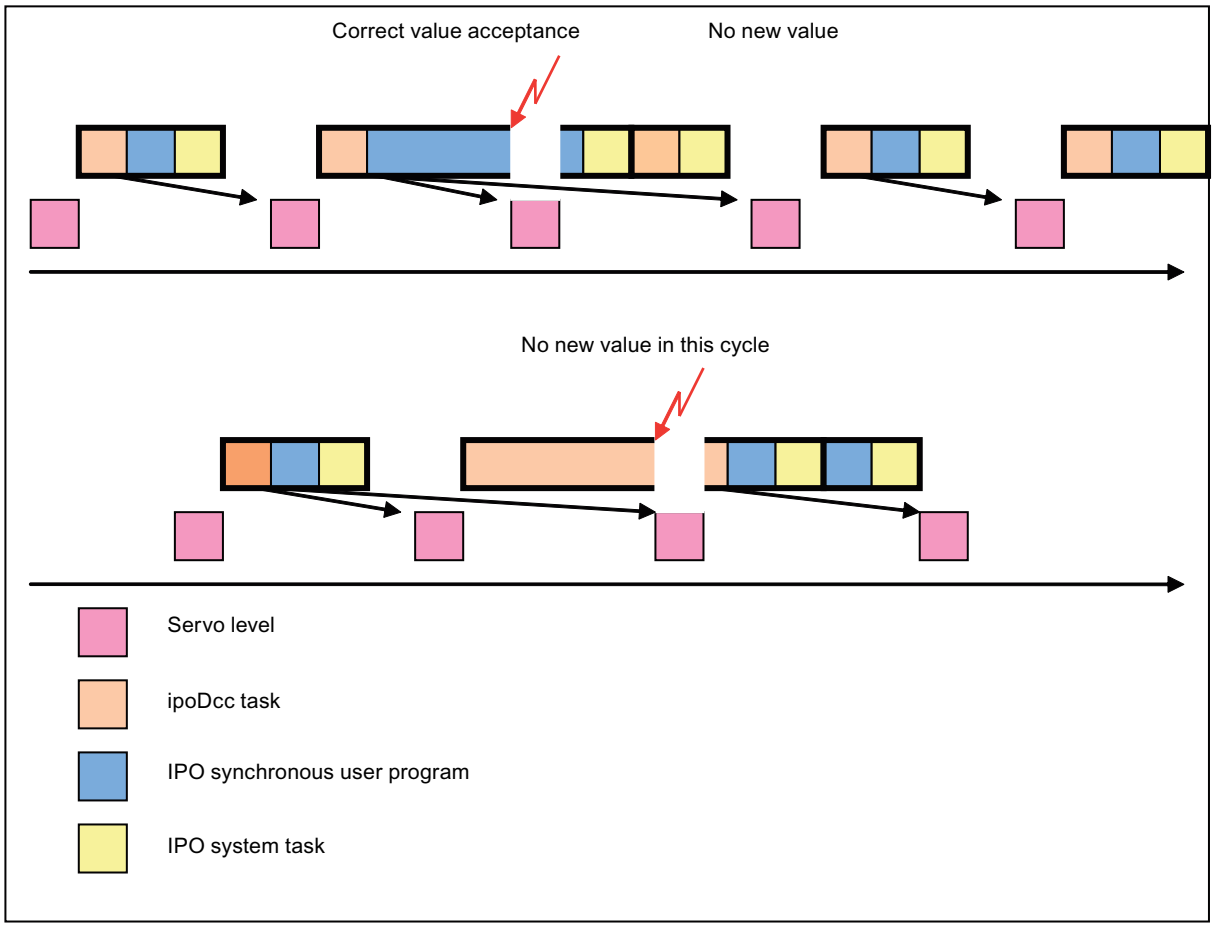


Figure 7-70 Overflow in the execution context with other tasks

Note

To ensure an isochronal data transfer between the ipoDCC task in the IPO level and the servoDCC in the servo level, the ipoDcc task in the IPO level considered by itself should not already cause a level overflow. If this is the case, the higher-priority task does not have available any updated values for the transfer of the values from the lower-priority level. The updated values are then transferred only after the next cycle of the lower-priority levels.

7.8.7.4 Data for blocks from a higher-priority level

Description

If an access is made to the output values from a lower-priority level, the values must be transferred from a dedicated cycle clock of the higher-priority level in order to guarantee the isochronism of input values.

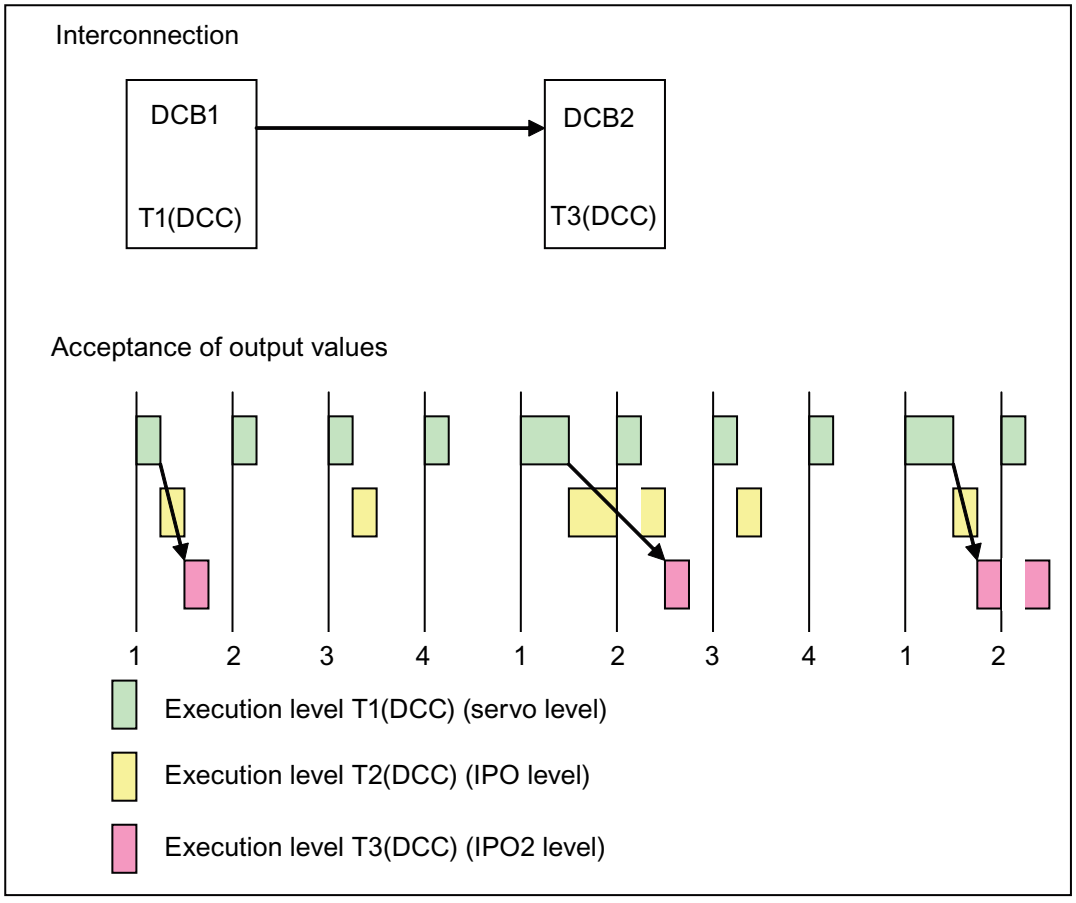


Figure 7-71 Data transfer from a higher-priority level

7.8.8 Interconnection of blocks with variables

7.8.8.1 Interconnection with variables

Overview of the interconnection possibilities with variables

In addition to the possibility of interconnecting blocks, a PIN can also be interconnected to an external variable. In this way, you can establish a connection with the I/O, the technology objects, a user program and an HMI.

The following user variables can be interconnected with blocks:

- **I/O variables**.
- **System variables**
- The following **user variables** can be interconnected with DCC:
  - Global device variables
  - Variables in the interface of a source

Interconnection of technology objects

- Interconnection using system variables and cyclic interface
- System functions of TOs cannot be called directly from DCCs.

Interconnect PIN with variables

You can interconnect the various inputs and outputs of a block in the DCC editor. The description of the DCC editor contains detailed information.

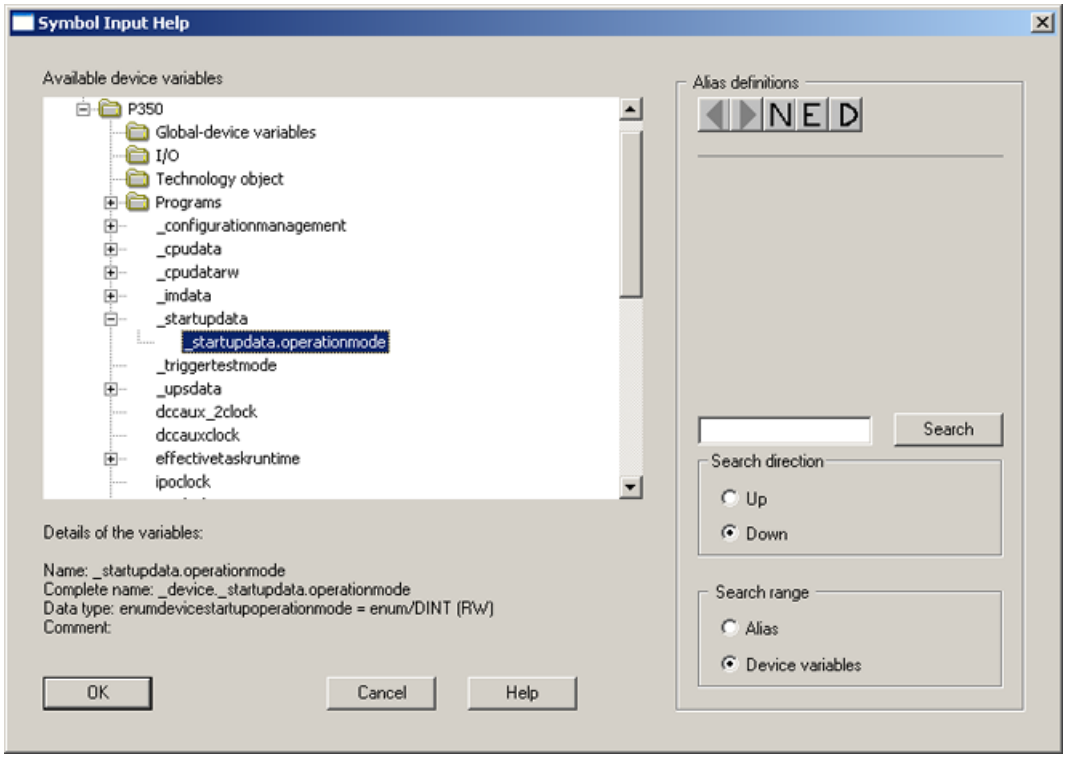


Figure 7-72 Interconnection of user variables in the DCC editor

Note

There are no restrictions as to how often an output can be interconnected to a variable. The output value of the most recently calculated block is always effective.

7.8.8.2 Behavior for FPU exceptions

Description

DCC Tasks exhibit the following behavior for FPU (Floating-Point Unit) exceptions:

Exception	Description	SIMOTION response
Invalid Operation	Invalid operation (e.g. 0/0, INF/ INF, INF - INF, SQRT(-1)	Device switches to STOP mode.
Division by Zero	Division by zero	Device switches to STOP mode.
Overflow	The absolute result of an operation is larger than abs(+/-- Nmax)	Device switches to STOP mode.
Underrun	The absolute result of an operation is less than abs(+/-- Nmax)	The result is 0.
Inexact	The result cannot be represented exactly.	The result is rounded.

See Error during operations with floating-point numbers (FPU exceptions) (Page 155).

7.9 Include drive I/O

7.9.1 Assigning the drive I/O symbolically

Description

As of V4.2, the drive I/O can be assigned symbolically to the I/O variables. For more information, refer to Assigning I/O variables via the address list (Page 100).

7.9.2 Terminal modules TM15 and TM17 High Feature

Description

The TM15 and TM17 High Feature terminal modules must be evaluated isochronously by the SIMOTION motion control system. Details for the timing can be obtained from the TM15 / TM17 High Feature Terminal Modules Commissioning Manual.

7.9.3 Terminal modules TMxx, TB30 and onboard I/Os on SIMOTION D or CX32/ CX32-2 and CU3xx/CU3xx-2

Description

TM31, TM41, TM15 DI/DO, TB30 and the onboard I/Os operate free-running (not isochronous) for SINAMICS.

The updating cycle is specified by the following drive parameters with 4 ms as default setting:

- p4099 for TM31, TM41 and TM15 DI/DO or
- p0799 for the onboard I/O

The accesses are performed in a SINAMICS background task (for example, every 4 ms). Access time to the inputs and outputs within the set updating cycle can vary from cycle to cycle; this means that the system merely ensures that within each cycle an update is made with a possible fluctuation range in accordance with the updating cycle. This can be influenced using the setting for p4099 or p0799.

Note

The following figure only provides a schematic presentation of the times and does not provide information on the absolute amount.

The I/O produces the following timing:

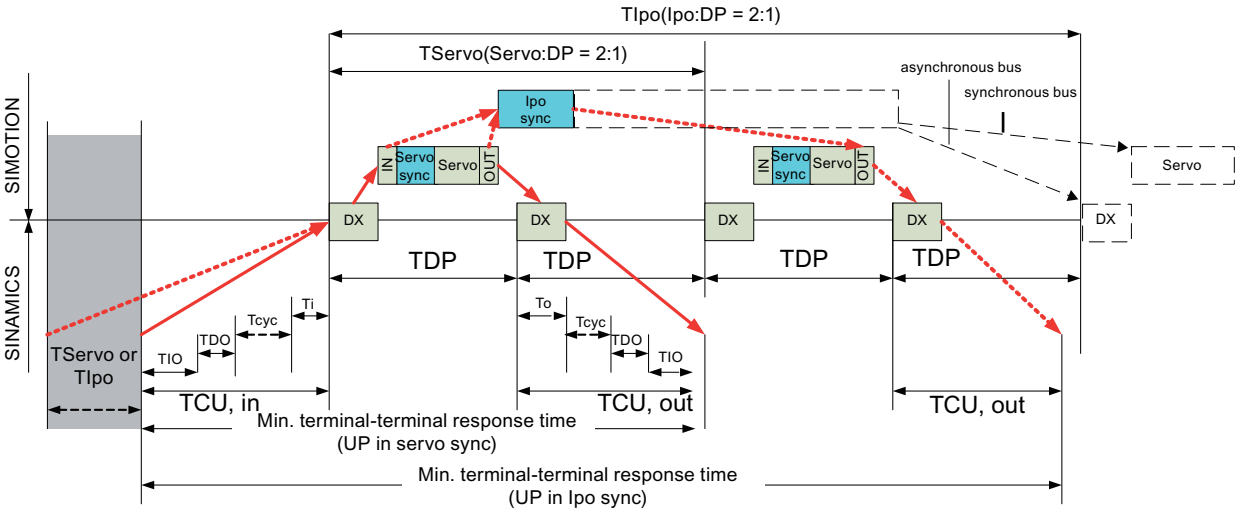


Figure 7-73 I/O Timing

Legend

- T_{DP}: Clock cycle PROFIBUS (see setting in HW Config)
- T_i: Latch time of the inputs for isochronous PROFIBUS (see setting in HW Config for the drive)
- T_o: Delay of the outputs for isochronous PROFIBUS (see setting in HW Config for the drive)
- T_{IO}: Module-specific signal delay (corresponds to a DRIVE-CLiQ cycle + the input delay time for digital input or the load-dependent output delay time for digital outputs)
- T_{cyc}: Updating cycle in the SINAMICS (p4099 or p0799 parameter)
- Servo: SIMOTION servo cycle clock; multiple of T_{DP} (see cycle clock settings for SIMOTION)
- IPO: SIMOTION IPO-cycle clock; multiple of the servo cycle clock (see cycle clock settings for SIMOTION)
- T_{DQ}: DRIVE-CLiQ clock cycle (for V4.1, always corresponds to the current controller cycle clock)
- UP: User program
- Output time on the CU: $T_{CU_out} = T_O + T_{cyc} + T_{DQ} + T_{IO}$
- Read in time on the CU: $T_{CU_in} = T_i + T_{cyc} + T_{DQ} + T_{IO}$

The figure shows all times that affect the terminal-terminal response time. Worst case values should be used for the dashed times, thus:

- The set updating cycle T_{cyc} (= maximum time for the access time)
- The time T_{servo} or T_{IPO} in the gray area. Depending on the time the input event occurs the maximum terminal-terminal reaction time can be up to one position control or IPO cycle clock longer than the minimum terminal-terminal reaction time.

The gray area shows the range in which the input event is acquired and processed with the next call of the user program (UP). The size of this range depends on whether the user program runs in the servo-synchronous task or IPO-synchronous task.

Note

Please note that a reduction of the update cycle leads to higher utilization of the SINAMICS closed-loop control or the SIMOTION D, which may result in a reduction in the quantity frame for drives, terminal modules, etc., under certain circumstances.

It is possible to access I/O components in the SINAMICS drive unit from the SIMOTION user program:

- As of V4.2, this can be carried out by means of symbolic assignment of the I/O variable to the SINAMICS I/O. SIMOTION automatically configures the required BICO interconnections and telegrams
- In V4.1, use can be made of this function with special telegrams (39x)
- In earlier versions, the I/O components are inserted in the PROFIdrive message frame as I/O PZD (process data) using a BICO interconnection (refer to the SIMOTION D Commissioning and Hardware Installation Manual, section "Message frame configuring for onboard I/O and drive objects").

The timing depends on which I/O component is accessed. The following table specifies the maximum delay times. To determine the terminal-terminal times, the values for the "terminal → user program" and "user program → terminal" must be added.

Note

The dashed line areas in the **IO Timing** graphic for the IPO synchronous task only apply for PROFINET IO. Analogously the times for PROFINET IO are presented in the following table in the **UP in IPOsynchronous task** column. For PROFIBUS DP in the line **OUT** T_{IPO} must be replaced by T_{Servo}. The dotted line area is irrelevant for PROFIBUS as the servo cannot run there over multiple bus cycles.

			Maximum delay times		
I/O module			**UP in servo-synchronous task**	**UP in iposynchronous task**	**T_{cyc}**
TM31, TM41, TM15 DI/O	Out	UP → terminal	$T_{DP} + T_{CU_out}$	$T_{IPO} + T_{DP} + T_{CU_out}$	p4099
	In	Terminal → UP	$T_{Servo} + T_{CU_in}$	$T_{IPO} + T_{CU_in}$	
TB30	Out	UP → terminal	$T_{DP} + T_o + T_{cyc} + T_{IO}$	$T_{IPO} + T_{DP} + T_o + T_{cyc} + T_{IO}$	p4099[3)]
	In	Terminal → UP	$T_{Servo} + T_i + T_{cyc} + T_{IO}$	$T_{IPO} + T_i + T_{cyc} + T_{IO}$	

			Maximum delay times		
I/O module			**UP in servo-synchronous task**	**UP in iposynchronous task**	**T_{cyc}**
Onboard I/O SIMOTION D, CX32, CX32-2, CU3xx, CU3xx-2	Out	UP → terminal[1)]	75 µs	75 µs	p0799[4)]
		UP → terminal[2)]	$T_{DP} + T_o + T_{cyc}$	$T_{IPO} + T_{DP} + T_o + T_{cyc}$	
	In	Terminal → UP	$T_{Servo} + T_i + T_{cyc}$	$T_{IPO} + T_i + T_{cyc}$	

1) Access to SIMOTION D integrated drives in conjunction with standard telegram 39x for the DO1. Without the standard message frame, the same timing behavior applies as for CX32 or CX32-2.

2.) Access to CX32, CX32-2 as well as via external PROFIBUS or PROFINET IO on CU3xx and CU3xx-2

3) For isochronous operation, T_{cyc} = max (T_{DP}, p4099)

4) For isochronous operation, T_{cyc} = max (T_{DP}, p0799)

Further information

You can also find more information on this topic:

- In the SIMOTION D410/D410-2 and D4x5/D4x5-2 Commissioning Manuals
- In an FAQ on the Utilities & Applications CD under FAQs
- On the Internet at http://support.automation.siemens.com/WW/view/de/27585482.

7.10 Time for SIMOTION and SINAMICS

7.10.1 SIMOTION time (real-time clock)

All SIMOTION devices have an integrated real-time clock. All events on a module (alarms, messages, etc.) are "time-stamped" based on the time shown by this real-time clock.

Setting the SIMOTION time of day

How to set the clock from SIMOTION SCOUT:

1. Select the SIMOTION device in the project navigator
2. Then select the **Target system > Set time of day** menu.

Alternatively, the clock can be set using the RTC system function block.

7.10.2 Synchronizing SINAMICS time of day

SINAMICS system runtime (operating hours counter)

With the SINAMICS Integrated of a SIMOTION D, controller extensions and the SINAMICS S120 control units, faults and warnings are "time-stamped" on the basis of the system runtime. This means that events are recorded by default on the basis of operating hours rather than a particular time of day or date.

System runtime

The entire system runtime is displayed in CU parameter r2114.

- r2114[0] shows the system runtime in milliseconds. After reaching 86,400,000 ms (24 hours), the value is reset.
- r2114[1] shows the system runtime in days.

The counter value is saved when the power is switched off. After the drive unit has been switched on, the counter continues to run with the value stored when the power was last switched off.

As a result, the drive displays the system runtime from 00:00:00 on 01.01.1992 in both the alarm window in SIMOTION SCOUT and the diagnostic buffer for entries.

If faults and warnings need to be "time-stamped" based on a time of day, "Time stamp operating hours" needs to be changed to "Time stamp UTC format" as described below.

Requirements

Telegram 39x is required for the time synchronization. If the automatic PROFIdrive telegram setting is selected for the control unit, this telegram is used automatically.

If the telegrams are specified manually, telegram 39x must be set up (see Section Telegram configuration (Page 110)).

So that drive units can be synchronized with the SIMOTION time, they must support telegram 39x and the UTC time format.

For precise time synchronization, the drive unit must also be operated via an isochronous/ equidistant bus on SIMOTION. The SINAMICS Integrated for SIMOTION D and the CX32/ CX32-2 controller extension always have an isochronous/equidistant connection.

The following control units support time synchronization:

- SINAMICS Integrated of the SIMOTION D
- CX32/CX32-2 controller extensions
- SINAMICS S120, CU310, CU310-2, CU320, CU320-2 control units, connected via PROFIBUS or PROFINET

Procedure

Proceed as follows to convert the SINAMICS clock to UTC format and to synchronize this with the SIMOTION clock:

1. Select the SIMOTION device in the project navigator.
2. Select the **Edit > Object Properties** menu.
3. Select the **Settings** tab.
4. Activate the option **Perform time synchronization with SINAMICS drive units**.
5. Confirm with OK.

Note

This setting is automatically activated for new projects as of V4.2 and applies for all drive units connected to the SIMOTION device. The SINAMICS clock is automatically synchronized with the SIMOTION clock for all drive units with configured telegram 39x.

The first time synchronization is performed after the SIMOTION device has reached the RUN operating state.

To compensate for deviations between the SIMOTION and SINAMICS clocks, the time of day is automatically resynchronized at regular intervals.

Using the system variable **_driveStates.allClocksSynchronized** on the SIMOTION device, the user program can query whether the automatic time synchronization is activated (=YES) or deactivated (=NO).

Before the first synchronization, alarms and messages are stored with the time stamp valid in the SINAMICS at this time, all subsequent alarms and messages with the synchronized time.

The first time synchronization after switching on is entered with the status of the operating hours counter and the time (UTC time, synchronized with SIMOTION) in the diagnostic buffer of the drive (e.g. SINAMICS Integrated).

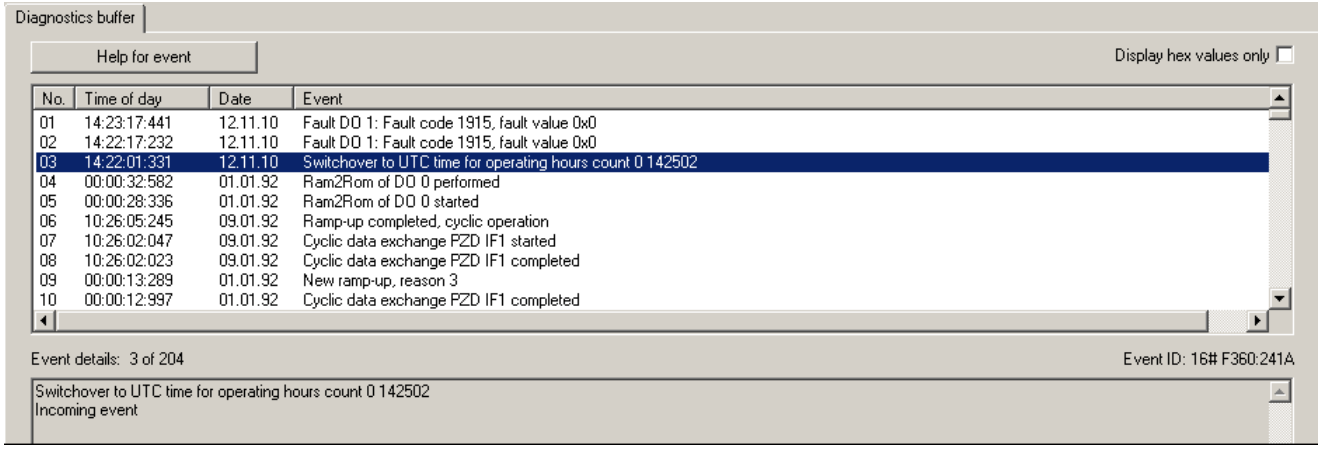


Figure 7-74 SINAMICS time synchronization diagnostic buffer entry

Error correction

If problems occur during the time synchronization, this may be due to faulty configuration information (Fast IO configuration) when symbolic assignment is deactivated.

For further information, see Section Message frame configuration (Page 110).

Compensation of runtime deviations

To compensate for deviations between the SIMOTION and SINAMICS clocks, the time of day is automatically resynchronized at regular intervals.

The following behavior must be taken into consideration when setting the SIMOTION time:

- "Time/date to be set" is after "Time/date on SINAMICS":
  Time and date are corrected on the SINAMICS.
- "Time/date to be set" is before "Time/date on SINAMICS":
  The SINAMICS clock must be stopped until the SINAMICS "Time/date" has caught up with "Time/date to be set".

Adopting this procedure ensures that the sequence of SINAMICS diagnostic buffer entries remains the same, even when the runtime differences are aligned.

The SINAMICS clock operates with a resolution of 1 ms. A synchronization accuracy of 1 ms can be achieved for all bus cycle clocks that can be divided exactly by 1 ms (e.g. 1 ms, 2 ms, 3 ms, etc.).

Due to system considerations, a slightly lower synchronization accuracy is achieved for all bus cycle clocks that cannot be divided exactly by 1 ms (e.g. 1.25 ms).

If the drive unit does not have an isochronous/equidistant connection on SIMOTION, this can result in runtime differences of milliseconds (depending on the set cycle clock ratios).

Resetting the time

Requirement:

- SIMOTION D4xx-2: As of SIMOTION V4.3
- SINAMICS S120: As of SINAMICS firmware version V4.5

A threshold value is defined using the CU parameter 3109 and is effective as follows:

- In the case of negative time jumps less than the threshold value (p3109), the time is halted (for details, see "Compensation of runtime deviations").
- In the case of negative time jumps greater than the threshold value (p3109), the time is reset.
  The factory setting of p3109 is 100 ms
  This means that in the case of negative time jumps greater than 100 ms, the time is reset.
  The factory setting is configured so that normal runtime deviations (drift of the quartzes) are below the threshold value.
  If the SIMOTION clock is reset by more than 100 ms, this is interpreted as a "specific reset of the time" and the time of the drives is also immediately reset.

If the real-time clock is reset by more than 60 seconds, a diagnostic buffer entry is also made in the drive:

Time correction (reset) by <correction value> seconds

After a re-synchronization (negative time gap greater than the threshold value), the CU parameter r3107 shows:

- In r3107[0..1] the UTC time after the synchronization
- In r3107[2..3] the UTC time before the synchronization

The milliseconds are in indices [0] and [2] and the days in [1] and [3].

Note

The diagnostic buffer entries are not converted to the new time when the time is changed.

Programming Execution System/Tasks/System Cycle Clocks

8

8.1 Execution system

8.1.1 Introduction to the execution system

All programs are executed in the SIMOTION device execution system. The execution system provides a series of execution levels with various execution properties.

Programs must therefore be assigned to the execution levels in order to be executed. To this end, programs of the source files are assigned to one or more tasks.

This defines the sequence in which they are to be executed.

Note

Before programs can be assigned to the execution levels the ST/MCC/LAD/FBD sources must be translated.

8.1.2 Execution levels and tasks

Execution levels define the chronological sequence of programs in the execution system. Each execution level contains one or more tasks.

A task provides the execution framework for the programs. You can assign one or more user programs to each task and specify their order within the task.

Besides user program tasks there are also several system tasks, the contents and execution sequence of which you cannot influence.

The table shows the execution levels with their tasks that are available for the user programs (user program tasks).

You can use task control commands to influence the execution system.

Table 8-1 Execution levels in SIMOTION

Execution level	Description
Time-controlled	Cyclic tasks, automatically restarted once assigned programs have been executed.
• SynchronousTasks	Tasks are started periodically, synchronous with specified system cycle clock. • ServoSynchronousTask: synchronous with the position control cycle clock (as of SIMOTION Kernel V4.0) • ServoSynchronousTask_fast: Synchronous with Servo_fast (Version 4.2 of SIMOTION Kernel and higher) as an option • IPOsynchronousTask: Synchronous with interpolator cycle clock IPO • IPOsynchronousTask_fast: Synchronous with interpolator cycle clock IPO_fast (Version 4.2 of SIMOTION Kernel and higher) as an option • IPOsynchronousTask_2: Synchronous with interpolator cycle clock IPO_2 • Synchronous tasks for the Tcontrol technology package: – PWMsynchronousTask: Synchronous to the PWM cycle clock InputSynchronousTask_1: Synchronous with cycle clock Input1 – InputSynchronousTask_2: Synchronous with cycle clock Input2 – PostControlTask_1: Synchronous with cycle clock Control1 – PostControlTask_2: Synchronous with cycle clock Control2
• TimerInterruptTasks	Tasks are started periodically in a fixed time frame. This time frame must be a multiple of interpolator cycle clock IPO.
Interrupts	Sequential tasks, executed once after start and then terminated.
• SystemInterruptTasks	Started when a system event occurs: Detailed description • ExecutionFaultTask: Error processing a program • PeripheralFaultTask: Error on I/O • TechnologicalFaultTask: Error on the technology object • TimeFaultBackgroundTask: BackgroundTask timeout • TimeFaultTask: TimerInterruptTask timeout
• UserInterruptTasks	Started on edge trigger when a user-defined event occurs.
Round robin	MotionTasks and BackgroundTasks share the free time remaining after execution of the higher-priority system and user tasks. The proportion of the two levels can be assigned.

Execution level	Description
• MotionTasks	Sequential tasks, executed once after start and then terminated. Start takes place: • Explicitly via a task control command in a program assigned to another task. • Automatically when RUN mode is attained if the corresponding attribute was set during task configuration. The priority of a MotionTask can be increased temporarily using the system function WAITFORCONDITION (see **Let MotionTask wait until a condition is satisfied**).
• BackgroundTask	Cyclic task, restarted automatically once the assigned programs have been executed; task cycle time depends on runtime.
StartupTask	Task is executed once when there is a transition from STOP or STOP U mode to RUN mode. SystemInterruptTasks are started by their triggering system event.
ShutdownTask	Task is executed once when there is a transition from RUN mode to STOP or STOP U mode. STOP or STOP U mode is reached by: • Corresponding mode selector switch position • Corresponding system function of the SIMOTION device • Alarm (fault) with corresponding error response SystemInterruptTasks and PeripheralFaultTasks are started by their triggering system event.
For information on behavior of sequential and cyclic tasks: • For the initialization of local program variables, refer to the Initialization of local program variables depending on the task execution behavior (Page 321) table. • In the event of execution errors in the program, see Processing errors in programs (Page 154) section. For information about options for accessing the process image and I/O variables, see the *Important properties for direct access and process image* section, which can be found in the various programming manuals.	

8.1.3 Task start sequence

When the StartupTask is completed, RUN mode is reached.

The following tasks are then started:

- SynchronousTasks
- TimerInterruptTasks
- BackgroundTask
- MotionTasks with startup attribute.

Note

The sequence in which these tasks are first started after RUN mode has been reached does not conform to the task priorities.

8.1.4 Configure execution system

8.1.4.1 Specifications for the configuring

When configuring the execution system, you assign the programs you want to execute in each task. In so doing, you specify the following:

- Priority with which the programs are executed
- Execution behavior (sequential, cyclical)
- Initialization behavior of local program variables

Assignment of a program to one or more tasks can only be performed after compilation and must occur before the program is loaded onto the target system.

When you have assigned a program to one or more tasks, you can establish the connection to the target system, download the program to the target system, and start it.

See also

Configure execution system (Page 244)

Assigning programs to the execution levels/tasks (Page 244)

8.1.4.2 Assigning programs to the tasks

In the *Execute sample program* section, you set up a task assignment using the sample program. The basic procedure is as follows:

1. Select the SIMOTION device in the project navigator, and select the **Target system > Configure execution system** menu command.
   The configuration window for the execution system of the SIMOTION device opens.
2. Select the task to be configured.
3. Select the **Program assignment** tab, and assign the desired programs to the task.
   See Section "Assigning programs to the execution levels / tasks (Page 244)".
4. Select the Task configuration tab and make any additional settings, such as:
   - The local data stack size in the Area limit for dynamic data field, see SIMOTION ST Programming Manual
   - Error response to program error
   - Time watchdogs for cyclic tasks
   - Start behavior of MotionTasks

   See the detailed description for the relevant task in Section "Description of the user program tasks (Page 212)".

8.1.5 Effect of the task execution behavior on the variable initialization

8.1.5.1 Time for the initialization of local program variables

The execution behavior of the task (sequential or cyclic) determines the initialization of local variables in the assigned programs. These descriptions also apply to instances of function blocks that were declared as local variables in the programs.

A summary of all variable types and their time of initialization can be found in the "Time of the variable initialization" chapter which can be found in the various Programming Manuals.

Table 8-2 Initialization of local program variables according to task execution behavior

Execution behavior	Tasks	Initialization of local program variables
Sequential (non-cyclic)	MotionTasks, UserInterruptTasks, SystemInterruptTasks, StartupTask, ShutdownTask.	After start, sequential tasks are executed once and then terminated. On each start, all local variables of the assigned programs are initialized. The time for data initialization is included in the task runtime. Behavior with compiler option "Only create program instance data once": • The local variables of the assigned programs will only be initialized once (during the download). See Download of changed sources in RUN (Page 568) and Influence of the compiler on variable initialization (Page 322). • Setting is possible so that the initialization of local variables is performed for every STOP - RUN transition, see Initialization of data during a STOP - RUN transition (Page 576).
Cyclic	BackgroundTask, SynchronousTasks, TimerInterruptTasks	Cyclic tasks are automatically restarted upon completion, and the values of static local variables of the assigned programs (declared in VAR/ END_VAR) are retained. • Static variables are initialized once only during the transition from STOP to RUN. • Temporary variables (declared in VAR_TEMP / END_VAR) are initialized every time the task is started. This one-time initialization of static variables is not included in the task runtime. Behavior with compiler option "Only create program instance data once": • The local variables of the assigned programs will only be initialized once (during the download). See Download of changed sources in RUN (Page 568) and Influence of the compiler on variable initialization (Page 322). • Setting is possible so that the initialization of local variables is performed for every STOP - RUN transition, see Initialization of data during a STOP - RUN transition (Page 576).

8.1.5.2 Assigning initial values to unit variables

Unit variables and global device variables are not initialized during the transition from STOP or STOPU mode to RUN mode (see the "Time of the variable initialization" chapter which can be found in the various Programming Manuals).

If you nevertheless want to assign initial values to these variables, use the StartupTask to do so. For information about the initialization for a STOP - RUN transition and the pragma BlockInit_OnDeviceRun (ST) or pragma lines (MCC and LAD/FBD), see also Initialization of data during a STOP - RUN transition (Page 576).

Note

The task start sequence is not defined after the transition to RUN mode (refer to the **Task start sequence** section).

Correct initial value assignment is not ensured if a task other than the StartupTask is used.

8.1.5.3 Use multiple VAR_GLOBAL, VAR_GLOBAL RETAIN blocks

Description

You can create multiple VAR_GLOBAL, VAR_GLOBAL RETAIN blocks in the interface and implementation area of a UNIT (as of V4.1).

In the interface area and in the implementation area you can specify multiple declaration blocks in any sequence. Each of these blocks will be versioned separately. Changes within a block (whether direct or indirect via data type change) thus result in reinitialization of these blocks when reloading. Thus the (RETAIN) data retain works block-by-block.

New blocks can be added on the end and the changed sources can be reloaded in the RUN, without influencing the existing data blocks. If in one source section (statement applies separately for interface and implementation) a block is added in front of an existing block of the same type (RETAIN or not RETAIN) then all the subsequent blocks change and will be reinitialized at download. Thus download in the RUN after such a change is not possible.

Via the functions _saveUnitDataSet /_loadUnitDataSet and _exportUnitDataSet / _importUnitDataSet, the unit data block information can be saved. If when reading a data set one or multiple blocks are missing this can be detected on the return code, the remaining blocks however will be read. See also General information on saving data sets from the user program (Page 427) and Data backup and data initialization from user program - functions and instructions (Page 481).

Loss of retain data due to initialization

This data can be saved first in SCOUT via the "Save variables" function and read in again via the "Restore variables" function, see Online help.

Alternatively you can also use the _exportUnitDataSet (Page 434) and _importUnitDataSet (Page 437) system functions in the application.

8.1.5.4 Influence of the compiler on variable initialization

Compiler option "Only create program instance data once"

The compiler option can be set on each source and thus overwrites the global setting. It works regardless of the creation language and consequently it can also be used in LAD/FBD and MCC.

The compiler option controls how the instance data must be created in a PROGRAM contained in a source. Instance data of a PROGRAM are formed by the content of the VAR declaration block.

Generally, the following is valid: A program can be hooked into the execution system once and only once per task.

If "Only create program instance data once" is not set (DEFAULT), then the following behavior applies:

- The program cannot be called from a different LAD.
- The instance data of a program will be created separately for each task to which the program is assigned.
- The data initialization of the instance data is executed with the start of the TASK; (sequential task with task start, cyclic tasks for the STOP-RUN transition)
- Instance data of PROGRAMs in sequential tasks can be changed in RUN mode (if a loading is possible in principle).

Activation of the compiler option "Only create program instance data once" (also when using a program in different tasks) causes the following:

- Instance data of programs compiled in this manner will only be created once. The instance data is in the source, where the PROGRAM is declared.
- Each time a PROGRAM created in this manner is used, the same data is involved. This affects the assignment to (possibly multiple) tasks and the call in other PROGRAMS or function blocks.
- The data initialization is executed according to the rules of global data initialization (see download settings on the project) with the download of the source / of the code in which the program is declared. See also Download of changed sources in RUN (Page 568).

NOTICE
Changed behavior for the data initialization when "Only create program instance data once" has been selected. With sequential tasks, data initialization is no longer performed with a task start or for cyclic tasks with the STOP - RUN transition, but generally only during a download. If necessary, the data must now be initialized by the application in the StartUp task or at the beginning of the programs in sequential tasks. An initialization during a STOP - RUN transition is possible • With a pragma BlockInit_OnDeviceRun (ST) or pragma line (MCC and LAD/FBD) • As of V4.2 using the dialog **Properties > Settings** via the context menu on the device. See Initialization of data during a STOP - RUN transition (Page 576). For information on the influence of the compiler option in conjunction with the download in RUN, see Download of changed sources in RUN (Page 568). If several tasks are assigned to one program, the same data is used as the basis for all of them.

Initialization of variables during a STOP - RUN transition

For information about the initialization for a STOP - RUN transition and pragma BlockInit_OnDeviceRun, see Initialization of data during a STOP - RUN transition (Page 576).

Compiler option "Permit language extensions" (for non-IEC conformity)

The compiler option can be set on each source and thus overwrites the global setting. It works regardless of the creation language and consequently it can also be used in LAD/FBD and MCC.

The compiler option allows the following:

- Direct bit access, bit addressing for bit string variables (except data type BOOL)
- Read and write access to the input parameter (VAR INPUT) from function blocks outside the function block.
- Permissible call "program in program".

A program can be called within a different program, like a global FB instance within a program organization unit, such as calling "myprog" within a different program, within a different FB, but not within a function.

Global availability of the instance data is a prerequisite for "program in program". This can be satisfied either in that the PROGRAM does not have any instance data, or by using the compiler option "Only create program instance data once" when compiling the PROGRAM that will be called (see **Only create program instance data once**).

Note

If you do not set the compiler options, then the behavior remains unchanged compared to V4.0 (corresponds to DEFAULT).

Re-initialization of variable blocks

In VAR_GLOBAL, VAR_GLOBAL RETAIN blocks (interface and implementation sections of the source/unit):

- `BlockInit_OnChange := true`; causes (IMPLEMENTATION only) a reinitialization of the data to be executed with the values specified in the source when changes are made to the block structure for the download in RUN.
- With MCC and LAD/FBD, as of V4.2 the behavior can be set in the declaration tables using pragma lines.

This pragma is also applicable in VAR declarations of PROGRAMs. However, it will only work if the "Only create program instance data once" compiler option is selected.

See also Download of changed sources in RUN (Page 568).

8.1.5.5 Marking HMI relevant data

As standard, the following applies for the HMI relevance of retentive unit variables (VAR_GLOBAL RETAIN) and non-retentive unit variables (VAR_GLOBAL):

- Unit variables declared in the interface section are available for operator control and monitoring (OCM).
  OCM addresses are generated for the variables contained therein. Changes have an effect on the HMI consistency.
- Unit variables declared in the implementation section are not exported to HMI devices. Changes have no effect on the HMI consistency.

Marking data as "non-HMI-relevant data"

You can mark individual declaration blocks in the interface section as non-HMI-relevant.

- In the SIMOTION ST programming language:
  Via the pragma `{ HMI_Export := FALSE; }` at the start of the declaration block (see the following example).
- In the SIMOTION MCC and SIMOTION LAD/FBD programming languages (as of V4.2):
  Via a pragma line in the declaration table.

OCM addresses are then no longer generated for the variables contained therein. For SIMOTION devices as of version V4.1, changes in this declaration block have no effect on the HMI consistency.

Example

```
VAR_GLOBAL
    { HMI_Export := FALSE; }
    x : INT;
    y : INT;
END_VAR
```

Marking data as "HMI-relevant data"

You can mark individual declaration blocks in the interface section as non-HMI-relevant.

- In the SIMOTION ST programming language:
  Via the pragma `{ HMI_Export := TRUE; }` at the start of the declaration block (see the following example).
- In the SIMOTION MCC and SIMOTION LAD/FBD programming languages (as of V4.2):
  Via a pragma line in the declaration table.

OCM addresses are then generated for the variables contained therein. Changes to this declaration block have an effect on the HMI consistency.

Example

```
VAR_GLOBAL
    { HMI_Export := TRUE; }
    x : INT;
    y : INT;
END_VAR
```

HMI address space and HMI consistency

The total size of the unit variables that can be exported to HMI devices is limited to 64 KB per unit.

Within the HMI address space, the variables are arranged in the following order:

- Retentive unit variables (VAR_GLOBAL RETAIN) of the interface section
- Retentive unit variables (VAR_GLOBAL RETAIN) of the implementation section
- Non-retentive unit variables (VAR_GLOBAL) of the interface section
- Non-retentive unit variables (VAR_GLOBAL) of the implementation section

Within these segments, the variables are arranged according to order of their declaration.

Note

For SIMOTION devices as of version V4.1:

Only HMI-relevant unit variables of the interface and implementation section are assigned to the HMI address space.

For SIMOTION devices up to version V4.0:

The following unit variables are assigned to the HMI address space:

- HMI-relevant **and non-HMI-relevant** unit variables of the interface section.
- HMI-relevant unit variables of the implementation section.

If variables of an HMI-relevant declaration block lie outside the HMI address space (64 KB), the compiler issues information 32050 that variables are not visible for the HMI. The first variable of the declaration block that exceeds the limit of the HMI address space is specified in this information.

Note

If variables are inserted in non-HMI-relevant declaration blocks, the HMI can still access SIMOTION variables even when the consistency check is activated.

A new compilation of the HMI project and a new transfer of the HMI project to an HMI device are then not necessary.

Exception for SIMOTION devices up to version V4.0:

HMI consistency is only retained when you enter or change variables in the following areas:

- In non-HMI-relevant declaration blocks of the **implementation section**
- In the **last declaration block** for non-retentive unit variables (VAR_GLOBAL) **of the interface section** when the following two conditions are satisfied:
  - This declaration block is marked as non-HMI-relevant.
  - No declaration blocks for non-retentive unit variables are marked as HMI-relevant in the implementation section.

  If the last condition is satisfied, you can add any number of declaration blocks for non-retentive unit variables (VAR_GLOBAL) at the **end of the interface section**, if they are marked as non-HMI-relevant.

See also HMI (Human Machine Interface) connection (Page 519) and HMI variables in one unit (Page 602).

For detailed information on the behavior of HMI-relevant data, see *SIMOTION ST Programming Manual*, Section Variables and HMI devices.

8.1.6 Task status

8.1.6.1 Querying and meaning of the task status

The task status can be queried using the function _getStateOfTaskId (Page 341). This function requires the *TaskId* as an input parameter and returns a value of data type DWORD.

The table shows the possible combinations of task statuses in hex representation and as symbolic constants. Task status combinations are possible and are displayed as the sum of the hexadecimal values.

Table 8-3 Task statuses and their meaning

Symbolic constant	Hex notation	Symbolic constant
TASK_STATE_INVALID	16#0000	The task does not exist.
TASK_STATE_STOP_PENDING	16#0001	Task has received signal to stop; it is in a state between TASK_STATE_RUNNING and TASK_STATE_STOPPED. Actions may be performed until the task has stopped.
TASK_STATE_STOPPED	16#0002	Task stopped (e.g. using _resetTaskID (Page 342) function), completed or not yet started.
TASK_STATE_RUNNING	16#0004	Task running, e.g.: • using _startTaskID (Page 345) function, • As an active cyclic task
TASK_STATE_WAITING	16#0010	Task waiting due to _waitTime function or WAITFORCONDITION.
TASK_STATE_SUSPENDED	16#0020	Task suspended by function _suspendTaskid (Page 346).
TASK_STATE_WAIT_NEXT_CYCLE	16#0040	TimerInterruptTask waiting for start trigger.
TASK_STATE_WAIT_NEXT_INTERRUPT	16#0080	SystemInterruptTask or UserInterruptTask is waiting for the triggering event to occur.
TASK_STATE_LOCKED	16#0100	Task locked by function *_disableScheduler*.

8.1.6.2 Combinations of task statuses

_getStateOfTaskId (Page 341) often delivers a return value in the form of combinations found in the *Keywords for the declaration of static and temporary variables* table, depending on the task status named by the source file section. These combinations are generated by an OR logic operation. Frequent combinations are listed in the table.

Table 8-4 Frequently occurring task status combinations

Combination	Hex notation	Meaning
TASK_STATE_WAITING OR TASK_STATE_RUNNING	16#0014	Task running but currently waiting, for example, because of *_waitTime* or WAITFORCONDITION.
TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#0024	Task running but currently suspended by _suspendTaskid (Page 346).
TASK_STATE_WAIT_NEXT_CYCLE OR TASK_STATE_RUNNING	16#0044	TimerInterruptTask running, but currently waiting for its start trigger.
TASK_STATE_WAIT_NEXT_CYCLE OR TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#0064	TimerInterruptTask running but currently waiting for its start trigger and suspended by _suspendTaskid (Page 346).
TASK_STATE_WAIT_NEXT_INTERRUPT OR TASK_STATE_RUNNING	16#0084	SystemInterruptTask or UserInterruptTask running, but currently waiting for its triggering event.
TASK_STATE_WAIT_NEXT_INTERRUPT OR TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#00A4	SystemInterruptTask or UserInterruptTask running, but currently waiting for its triggering event, and is suspended by *_suspendTaskid*.
TASK_STATE_LOCKED OR TASK_STATE_RUNNING	16#0104	Task running but currently locked by *_disableScheduler*.

Other combinations are possible.

8.1.6.3 Example of the task status in use

In the following example, the task status of a MotionTask is queried to decide whether it can be started. The relevant bits of the return value are evaluated for this. If the result is positive, the MotionTask is started.

Table 8-5 Example of a query whether a MotionTask can be started

```
ret_dword := _getStateOfTaskId (id := _task.motionTask_1);
    IF (ret_dword AND
        (TASK_STATE_STOPPED OR TASK_STATE_STOP_PENDING)
        ) <> 0 THEN
        // MotionTask can be started.
        ret_dword := _restartTaskId (id := _task.motionTask_1);
    ELSE
        ; // MotionTask cannot be started.
    END_IF;
```

For an example with MCC: See Section "Parameter overview for the task state" in the MCC Programming Manual.

8.1.7 Waiting in MotionTask for a condition to be satisfied

8.1.7.1 Syntax of the condition of the EXPRESSION

The following applies for programming in ST. For MCC, definition of the wait condition is executed directly in the commands.

You can use the WAITFORCONDITION command in a MotionTask to wait for a condition to be satisfied (e.g. an event to occur). The MotionTask in which the statement is called is kept in TASK_STATE_WAITING status until the condition is satisfied.

When the VAR_IN_OUT in/out parameter is used, you can enable a time watchdog for WAITFORCONDITION.

This condition is formulated in the form of an EXPRESSION.

The expression is a special type of function declaration (for information on the syntax, see the **Expressions** section in the ST Programming Manual):

- The data type of the return value can be defined as BOOL and is not specified explicitly.
- The use of the VAR_IN_OUT in/out parameter and the VAR_INPUT input parameter is possible.

An expression can only be declared in the implementation section of the unit.

Optionally, local (temporary) variables can be declared in the declaration section. No other declarations (e.g. of input parameters or constants) are possible.

The following can be accessed in the statement section:

- Local variables for the expression
- Unit variables
- Global device variables, I/O variables, and the process image

An expression of the BOOL data type must be assigned to the expression identifier in the statement section of the expression (see **Expressions** section in the ST Programming Manual).

Note

The statement section of the expression cannot contain any function calls or loops.

8.1.7.2 Syntax of the WAITFORCONDITION statement

The following syntax is used to call the command for a waiting MotionTask subject to an expression:

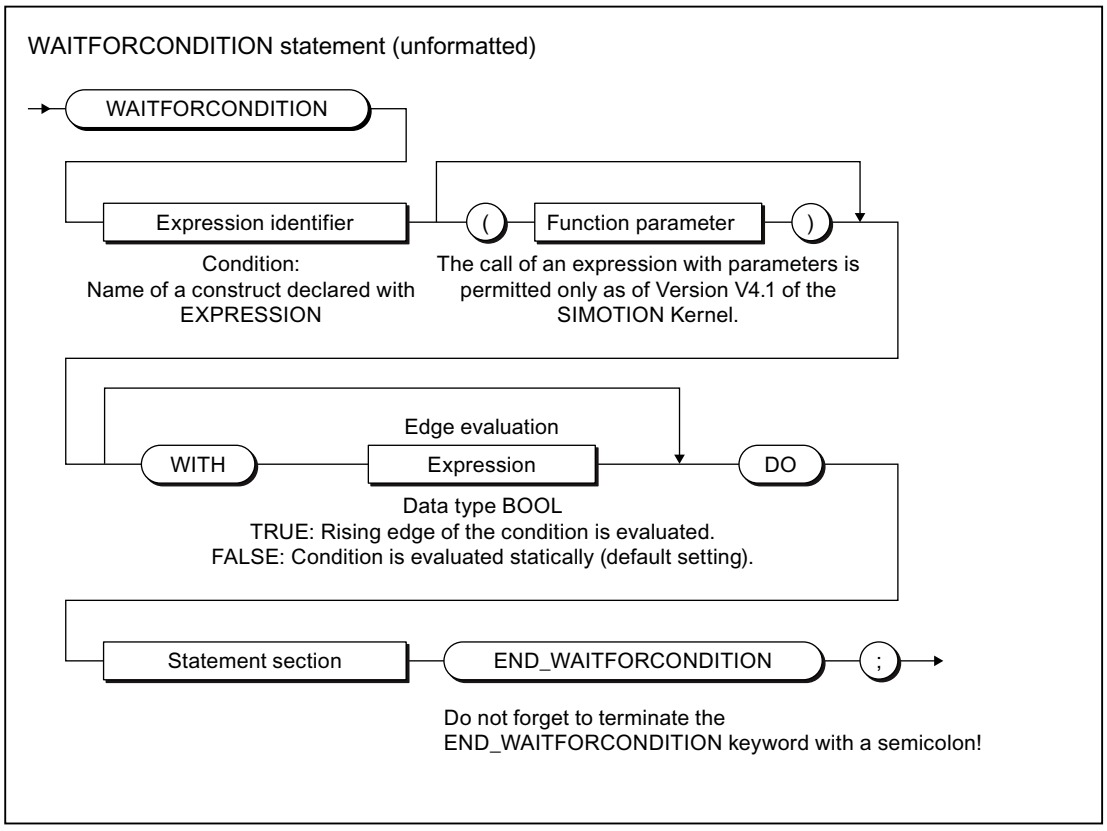


Figure 8-1 Syntax: WAITFORCONDITION statement

Expression identifier is a construct declared with EXPRESSION; its value defines (together with *WITH edge evaluation*, if necessary) whether the condition is considered as satisfied.

The *WITH edge evaluation* sequence is optional. *Edge evaluation* is an expression of data type BOOL; it determines how the value of *expression identifier* is interpreted:

- *Edge evaluation* = TRUE: The rising edge of *expression identifier* is interpreted; i.e. the condition is satisfied when the value of *expression identifier* **changes** from FALSE to TRUE.
- *Edge evaluation* = FALSE: The static value of *expression identifier* is evaluated; i.e. the condition is satisfied when the value of *expression identifier* **is** TRUE.

If is not specified, the default setting is FALSE, i.e. the static value of *expression identifier* is evaluated.

The statement section must contain at least one statement (empty statements also possible).

8.1.7.3 Effect of the WAITFORCONDITION statement

The WAITFORCONDITION statement sets the task that called it to the TASK_STATE_WAITING state until the condition (expression) is TRUE.

The priority of the task is increased if the expression is true. The priority is higher than for UserInterruptTasks and lower than for SystemInterruptTasks, i.e. the MotionTask has its turn before the other task in the round-robin execution level, as well as before the UserInterruptTasks and TimerInterruptTasks.

The increased task priority applies to all statements between the WAITFORCONDITION command and the END_WAITFORCONDITION command. The END_WAITFORCONDITION command ends the increased task priority.

The statement section must contain at least one empty statement.

Please note the following when using the WAITFORCONDITION command:

- The command is used to wait for an event in a MotionTask. If it is programmed within another task, it is ignored.
- There is no time watchdog in a MotionTask.
  Therefore, make sure that the condition does actually become true. Otherwise, the MotionTask would always be in the wait state and this would produce a timeout error.
- As of V4.1 a time watchdog for WAITFORCONDITION can be programmed in a Motion Task.
- The WAITFORCONDITION ... END_WAITFORCONDITION structure may not be nested.
- Within the round-robin level, the waiting task is the next to be executed when the event occurs, as long as it is not hindered by higher-priority tasks (such as alarms).
- The time slice of the active task in the round robin level is interrupted.
- For the behavior in the event of task interruption, see *_suspendTaskid*).
- The expression is checked in the IPO cycle clock with high priority.

See also

Task priorities (Page 205)

MotionTasks (Page 215)

8.1.7.4 Example of use of WAITFORCONDITION

The following example assumes that the *feeder* program is running in a MotionTask. The option **Activation after StartupTask** is selected for this MotionTask.

The assignment of programs to tasks is performed in SIMOTION SCOUT (see **Assigning programs to the execution levels / tasks**).

Table 8-6 Example for use of the WAITFORCONDITION condition

```
INTERFACE
 USEPACKAGE CAM;
 PROGRAM feeder;
END_INTERFACE

IMPLEMENTATION
    // Condition for WAITFORCONDITION in MotionTask_1
    EXPRESSION automaticExpr
        automaticExpr := IOfeedCam; // Digital input
    END_EXPRESSION
    // feeder (MotionTask_1)
    PROGRAM feeder
        VAR
            retVal : DINT;
    END_VAR ;
    retVal := _enableAxis(axis := realAxis,
        enableMode := ALL,
        servoCommandToActualMode := INACTIVE,
        nextCommand := WHEN_COMMAND_DONE,
        commandId := _getCommandId() );
        // Wait for automatic start
    WAITFORCONDITION automaticExpr WITH TRUE DO
        retVal := _pos(axis := realAxis,
            positioningMode := RELATIVE,
            position := 500,
            velocityType := DIRECT,
                                velocity:=300,
            positiveAccelType := DIRECT,
            positiveAccel:= 400,
            negativeAccelType := DIRECT,
            negativeAccel := 400,
            velocityProfile:= TRAPEZOIDAL,
                                mergeMode:=IMMEDIATELY,
                                nextCommand:=WHEN_MOTION_DONE,
            commandId:= _getCommandId() );
            // Reduce priority after WAITFORCONDITION
    END_WAITFORCONDITION;
    retVal := _disableAxis(axis := realAxis,
    disableMode := ALL,
    servoCommandToActualMode := INACTIVE,
    nextCommand := WHEN_COMMAND_DONE,
    commandId := _getCommandId() );
    END_PROGRAM
END_IMPLEMENTATION
```

8.1.7.5 Example for time verification via FB

Example of using WAITFORCONDITION with time verification via FB (as of V4.1)

The following example shows time monitoring of the *expressions* (WAITFORCONDITION). The bonding of the expression at the call point to instance data of a calling function block inclusive time monitoring is presented alongside. This is a TON type reference variable. The call is executed within the expression with query of the output.

```
INTERFACE
    VAR_GLOBAL
        v1, v2 : INT;
        t1, t2 : TIME;
        Evaluation: INT;
    END_VAR
    PROGRAM Test_1;
END_INTERFACE
IMPLEMENTATION
    EXPRESSION mcccont1
        VAR_IN_OUT
            v : INT;
            t : TON;
        END_VAR
        t();
        mcccont1 := v > 100 or t.q ;
    END_EXPRESSION
    FUNCTION_BLOCK waitfb
        VAR_IN_OUT
            refpar1 : INT;
            refpar2 : TIME;
        END_VAR
        VAR_TEMP
            expr_timeout : TON;
        END_VAR
        expr_timeout(pt := refpar2, IN := TRUE);
        // Set monitoring time and
        //activate timer
        Evaluation: = 0;
        WAITFORCONDITION mcccont1(v := refpar1, t := expr_timeout ) DO
           Evaluation: = 100; //statement
            IF (expr_timeout.q) THEN
                // Error handling feasible, if Time-Out
                Evaluation: = 20005;
            END_IF;
        END_WAITFORCONDITION;
        expr_timeout( IN := FALSE);
        ;
    END_FUNCTION_BLOCK
    PROGRAM test_1
        VAR
            my_waitfb1 : waitfb;
        END_VAR
        my_waitfb1 (refpar1 :=v1, refpar2:=t1);
        //Wait until V1>100 and time
    END_PROGRAM
END_IMPLEMENTATION
```

The call of the waitfb type instance can then be executed at any point with different variables in each case. These global variables are then updated by cyclic tasks:

```
my_waitfb_1(refpar1 := v1, refpar2:=t1); //Wait until V1>100 and
time t1 not elapsed

my_waitfb_2(refpar1 := v2, refpar2:=t1);
```

8.1.7.6 Example for using WAITFORCONDITION with time monitoring directly in non-cyclic task / Motion Task

Description

Example of time monitoring of the expression (WAITFORCONDITION).

The call is executed directly in a MotionTask.

The time monitoring is type TON, with call within the expression and query of the output.

```
INTERFACE
    TYPE
        eDiagType : (TIMEOUT, COUNTER_REACHED, NOTHING);
    END_TYPE
    VAR_GLOBAL
        eDiag   : eDiagType := NOTHING;
        v1      : INT  := 0;
        t1      : TIME := T#0d_0h_0m_3s_0ms;
    END_VAR
    PROGRAM testExpression;
    PROGRAM increaseV1;
END_INTERFACE
IMPLEMENTATION
    EXPRESSION expr
        VAR_IN_OUT
            v : INT;
            t : TON;
        END_VAR
        t();
        expr := v > 500 OR t.q;
    END_EXPRESSION
                         //MotionTask
    PROGRAM testExpression
        VAR_TEMP
            expr_timeout : TON;
        END_VAR
                        // Set monitoring time, and activate timer
expr_timeout(pt := t1, IN := TRUE);
                       //Wait until v1 > 500 or time t1 has elapsed
        WAITFORCONDITION expr(v := v1, t := expr_timeout) DO
            IF (expr_timeout.q) THEN
                       // Error handling
                       // Time t1 has elapsed without v1 > 500 becoming true
                eDiag := TIMEOUT;
                … ;
            ELSE
                       // Good case,  v1 > 500
                eDiag := COUNTER_REACHED;
                 … ;
            END_IF;
        END_WAITFORCONDITION;
    END_PROGRAM
                         //BackgroundTask
    PROGRAM increaseV1
        v1 := v1 + 1;
    END_PROGRAM
END_IMPLEMENTATION
.....
```

You can update v1 in any cyclical tasks.

8.1.8 Making tasks wait a defined period

Place task in the wait state

You can place a task in the TASK_STATE_WAITING status for a specific period of time (see Task status (Page 327)). To do so, use the *_waitTime* function. This places the calling task in the wait state for the specified period.

Note

A task that is in the wait state does not (or hardly) requires any CPU time. The only load on the target system is the periodic check to establish whether the wait time has expired. This check takes place in the IPO cycle clock.

The call of *_waitTime (timeValue := T#0ms)* in a MotionTask temporarily inactivates these and returns the program control to the scheduler. This is recommended, for example for longer loops if the program control will knowingly be transferred to the next round robin task (also possible in BackgroundTask).

Note

The function should be used in MotionTasks only; using it in cyclic tasks may lead to time monitoring errors!

- With SynchronousTasks: You can configure whether the time watchdog is suspended. Time monitoring is active by default.
  With IPOsynchronousTask, additionally take the following into account: UserInterruptTasks will no longer be started by their triggering event!
- With other cyclic tasks (BackgroundTask, TimerInterruptTasks): Time monitoring is always active.

In cyclic tasks, use the timer system function blocks (see *Timers* section) to implement waiting times.

Use an expression of data type TIME as an input parameter, the return value of data type DINT is always 0.

For more information on the function (syntax), refer to the description of the **_waitTime** function.

The following sample program puts the MotionTask to which it is assigned in the wait state for ten seconds:

Table 8-7 Example of _waitTime function

```
INTERFACE
    PROGRAM waitTime;
END_INTERFACE

IMPLEMENTATION
    PROGRAM waitTime
        VAR
            retVal :DINT;
        END_VAR;
        retVal := _waitTime(timeValue := T#10s);
    END_PROGRAM
END_IMPLEMENTATION
```

8.2 Task control commands

8.2.1 Overview of the task control commands

The commands listed in the table are available for task control.

Table 8-8 Task control commands in SIMOTION ST

Task control command	Meaning
_startTaskId (Page 345)	Starts a MotionTask that is in the state TASK_STATE_STOPPED; its startup code is executed. Can only be applied to MotionTasks. Command must not directly follow *_resetTaskId*. Use *_restartTaskId* instead.
_resetTaskId (Page 342)	Resets a MotionTask to TASK_STATE_STOPPED. Can only be applied to MotionTasks. *_startTaskId* must not directly follow this command. Use *_restartTaskId* instead.
_restartTaskId (Page 343)	Resets a MotionTask to TASK_STATE_STOPPED and restarts it; the startup code for this MotionTask is executed. Can only be applied to MotionTasks.
_suspendTaskId (Page 346)	The task in question will be suspended (set to TASK_STATE_SUSPENDED), for cyclic tasks time monitoring will be suspended. Cannot be used on SynchronousTasks, StartupTask, or ShutdownTask.

Task control command	Meaning
_resumeTaskId (Page 343)	The relevant task, which is located in the TASK_STATE_SUSPENDED state, is resumed. For cyclic tasks, time monitoring is activated again. Cannot be used on SynchronousTasks, StartupTask, or ShutdownTask.
_getStateOfTaskId (Page 341)	Queries the state of the relevant task.
_retriggerTaskIdControlTime (Page 344)	The monitoring time is reset once for the relevant task.

The task is specified for these functions via a unique TaskId (Page 339).

An example of a task control command can be found in Section "Example for using a task control command (Page 340)".

See the SIMOTION MCC Programming Manual (Chapters "Incoming message" and "Outgoing message") for programming functions in MCC.

See also Task status (Page 327).

Commands for controlling the scheduler

In addition, SIMOTION devices provide system functions for controlling the scheduler. These are listed in the table (for a description, see the List Manuals for SIMOTION devices):

Table 8-9 Commands for controlling the scheduler

Task control command	Meaning
_disableScheduler()	Prevents all user program tasks (except SynchronousTasks) from being loaded until *_enableScheduler* is called. It does not, however, affect system tasks. **Note**: The time monitoring for cyclic tasks is **not** suspended. **Note**: This command also prevents exchanging of SystemInterruptTasks and UserInterruptTasks!
_enableScheduler()	Cancels the effect of *_disableScheduler*.

8.2.2 TaskId

Tasks are specified via the associated TaskId.

You can obtain the TaskId for a task (data type StructTaskId) as follows:

- As _task.*name* variable (e.g. _task.motionTask_1). Explanation of terms:
  - _task
    Predefined name space for TaskId, see Section "Name spaces" in the SIMOTION ST Programming Manual. The identifier for the name space is separated by a period from the following identifier.
  - *name*
    The identifier of the task as specified in the execution system (e.g. motionTask_1).
- With the _getTaskId (*name*) function, see _getTaskId (Page 347) function.
  name is the identifier of the task as assigned in the execution system.

You can use the _checkEqualTask function to check whether a TaskId belongs to a particular task (see _checkEqualTask (Page 348) function).

Note

The _checkEqualTask and _getTaskId functions may only be called in the short form, i.e. with a complete listing of all parameter values, but without specification of the formal parameters.

The _checkEqualTask and _getTaskId functions and variables of the form _task.*name* must not be used in libraries.

8.2.3 Example for using a task control command

The following example requires that the following assignments have been made in SIMOTION SCOUT:

- The *myStart* program is assigned to the StartupTask.
- The *myMotion* program is assigned to MotionTask_1.

After controller startup (transition from STOP to RUN), the StartupTask runs automatically. The *myStart* program is executed in this task.

The *_startTaskId* task control command is used in the *myStart* program to start MotionTask_1. In MotionTask_1, the *myMotion* program is executed.

```
INTERFACE
    PROGRAM myStart;
    PROGRAM myMotion;
END_INTERFACE

IMPLEMENTATION
    PROGRAM myStart
        VAR
            RetDWord : DWORD;
        END_VAR

        RetDWord := _startTaskid (_task.motionTask_1);
        // Start MotionTask_1
    END_PROGRAM

    PROGRAM myMotion
        ;// Here, commands in the program, e.g. axis commands
    END_PROGRAM
END_IMPLEMENTATION
```

8.2.4 _getStateOfTaskId function

Description

This functions returns the state of the relevant task. The task is specified by means of a project-wide unique TaskId (see TaskId (Page 339)).

The function can be applied to all tasks except StartupTask and ShutdownTask.

See also Section "Task status (Page 327)". An example is also provided of how to decide if a MotionTask can be started, based on the task status, see Example for use of the task status (Page 328).

Declaration

```
_getStateOfTaskId (
             { id     : StructTaskId // TaskId
             }
             ) : DWORD
```

Input parameter

id (optional)

Data type: StructTaskId

Default: TaskId of the current task in which the function is called.

TaskId of the task to be controlled, see TaskId (Page 339).

Return value

Data type: DWORD

Status of the task is displayed as an OR operation on the following values, see also Task status (Page 327):

16#0000	Task does not exist or Task ID is invalid TASK_STATE_INVALID
16#0001	Task in transition from running to stopped TASK_STATE_STOP_PENDING
16#0002	Task stopped TASK_STATE_STOPPED
16#0004	Task running TASK_STATE_RUNNING
16#0010	Task waiting TASK_STATE_WAITING
16#0020	Task suspended TASK_STATE_SUSPENDED
16#0040	TimerInterruptTask waiting for its start trigger TASK_STATE_WAIT_NEXT_CYCLE

16#0080	SystemInterruptTask or UserInterruptTask waiting for the occurrence of an initiating event TASK_STATE_WAIT_NEXT_INTERRUPT
16#0100	Task locked (by _disableScheduler) TASK_STATE_LOCKED

8.2.5 _resetTaskId function

Description

The function resets a MotionTask to TASK_STATE_STOPPED. The task is specified by means of a project-wide unique TaskId (see TaskId (Page 339)).

The function is only applicable to MotionTasks.

The _startTaskId (Page 345) function must not directly follow this function. Use the _restartTaskId (Page 343) function instead.

Declaration

```
_resetTaskId (
         id    : StructTaskId    // MotionTasks only
         ) : DWORD
```

Input parameter

id

Data type: StructTaskId

For the TaskId of the task to be controlled, see TaskId (Page 339) – MotionTasks only.

Return value

Data type:	DWORD
0	No error
16#FFFF_FFFE	TaskId does not refer to a MotionTask
16#FFFF_FFFF	TaskId is invalid

8.2.6 _restartTaskId function

Description

The function resets a MotionTask to TASK_STATE_STOPPED and restarts it; its data is initialized. The task is specified by means of a project-wide unique TaskId (see TaskId (Page 339)).

The function is only applicable to MotionTasks.

To prevent this function from being used to stop a running MotionTask, its status can be queried and evaluated, see _getStateOfTaskId (Page 341) function and the Example for use of the task status (Page 328).

Declaration

```
_restartTaskId (
         id    : StructTaskId  // MotionTasks only
         ) : DWORD
```

Input parameter

id

Data type: StructTaskId

For the TaskId of the task to be controlled, see TaskId (Page 339) – MotionTasks only.

Return value

Data type:	DWORD
0	No error
16#FFFF_FFFE	TaskId does not refer to a MotionTask
16#FFFF_FFFF	TaskId is invalid

8.2.7 _resumeTaskId function

Description

The function resumes a task that is in the TASK_STATE_SUSPENDED state. This state has been achieved using the _suspendTaskId (Page 346) function. The task is specified by means of a project-wide unique TaskId (see TaskId (Page 339)).

The function is applicable to MotionTasks, BackgroundTask, TimerInterruptTasks, UserInterruptTasks, SystemInterruptTasks.

In the case of cyclic tasks (BackgroundTask, TimerInterruptTasks), the time watchdog for the task is enabled again.

Declaration

```
_resumeTaskId (
         id    : StructTaskId
         ) : DWORD
```

Input parameter

id

Data type: StructTaskId

TaskId of the task to be controlled, see TaskId (Page 339).

Return value

Data type:	DWORD
0	No error
16#FFFF_FFFE	TaskId refers to an illegal task
16#FFFF_FFFF	TaskId is invalid

8.2.8 _retriggerTaskIdControlTime function

Description

The function resets the monitoring time once for the relevant task. The task is specified by means of a project-wide unique TaskId (see TaskId (Page 339)).

The function is applicable to MotionTasks, BackgroundTask, TimerInterruptTasks, UserInterruptTasks, SystemInterruptTasks.

Declaration

```
_retriggerTaskIdControlTime (
                     { id        : StructTaskId
                     }
                  ) : DWORD
```

Input parameter

id (optional)

Data type: StructTaskId

Default: TaskId of the current task in which the function is called.

TaskId of the task to be controlled, see TaskId (Page 339).

Return value

Data type:	DWORD
0	Function executed normally.
Not equal to 0	Function not executed normally.

8.2.9 _startTaskId function

Description

The function starts a MotionTask that is in the TASK_STATE_STOPPED state; its data is initialized. The task is specified by means of a project-wide unique TaskId (see TaskId (Page 339)).

The function is only applicable to MotionTasks.

It has no effect on MotionTasks in the following states:

- TASK_STATE_RUNNING
- TASK_STATE_WAITING
- TASK_STATE_SUSPENDED

Before using the function, the state of the MotionTask can be queried and evaluated, see _getStateOfTaskId (Page 341) function and Example for use of the task status (Page 328).

It must not directly follow the _resetTaskId (Page 342) function. Use the _restartTaskId (Page 343) function instead.

Declaration

```
_startTaskId (
         id    : StructTaskId    // MotionTasks only
         ) : DWORD
```

Input parameter

id

Data type: StructTaskId

TaskId of the task to be controlled, see TaskId (Page 339) – MotionTasks only.

Return value

Data type:	DWORD
0	No error
16#FFFF_FFFE	TaskId does not refer to a MotionTask
16#FFFF_FFFF	TaskId is invalid

8.2.10 _suspendTaskId function

Description

The function suspends the relevant task (sets it to TASK_STATE_SUSPENDED). The task is specified by means of a project-wide unique TaskId (see TaskId (Page 339)).

The function is applicable to MotionTasks, BackgroundTask, TimerInterruptTasks, UserInterruptTasks, SystemInterruptTasks.

In the case of cyclic tasks (BackgroundTask, TimerInterruptTasks), the time watchdog for the task is stopped.

The task and its time monitoring are resumed using the _resumeTaskId (Page 343) function.

Declaration

```
_suspendTaskId (
              id         : StructTaskId
          ) : DWORD
```

Input parameter

id

Data type: StructTaskId

TaskId of the task to be controlled, see TaskId (Page 339).

Return value

Data type:	DWORD
0	No error
16#FFFF_FFFE	TaskId refers to an illegal task
16#FFFF_FFFF	TaskId is invalid

Interruption of the evaluation of a condition for WAITFORCONDITION

When the wait condition of a WAITFORCONDITION statement in a MotionTask is checked, the task status is not taken into account automatically.

You have programmed the wait condition for WAITFORCONDITION statement, e.g. wait for overtravel of an initiator, in an EXPRESSION. The associated MotionTask is now interrupted with _suspendTaskId before the wait condition is satisfied. Subsequently the condition is temporarily satisfied, e.g. through manual overtravel of an initiator. After _resumeTaskId, the MotionTask is executed as if the condition had been satisfied in the normal process.

However, you can avoid this by additionally querying the task status in the EXPRESSION.

```
// Condition for WAITFORCONDITION statement in MotionTask_1
EXPRESSION automaticExpr
    automaticExpr := IOfeedCam AND        // Digital input
        (task_status = (TASK_STATE_RUNNING OR TASK_STATE_WAITING));
END_EXPRESSION
```

8.2.11 _getTaskId function

Description

The function generates a project-wide unique TaskId (Page 339) from the task name (as in the execution system). This TaskId can be assigned to a variable of data type StructTaskId and used as an input parameter in the following functions:

- Task control commands (Page 338)
- Functions for runtime measurement of tasks (Page 349)

Note

This function must not be used in libraries.

This function may only be called in the short form, i.e. with a complete listing of all parameter values, but without specification of the formal parameters.

Declaration

```
_getTaskId (  // Only the short form is permitted
         name    : Task_Name  // Name of the task
         ) : StructTaskId
```

Input parameter

name (only short form permitted)

Name of task as defined in the execution system.

Return value

Data type: StructTaskId

Return value contains the TaskId of the task.

8.2.12 _checkEqualTask function

Description

This function indicates whether a TaskId (Page 339) is associated with a specified task.

Note

This function must not be used in libraries.

This function may only be called in the short form, i.e. with a complete listing of all parameter values, but without specification of the formal parameters.

Declaration

```
_checkEqualTask (  // Only the short form is permitted
        id     : StructTaskId,  // TaskId
        Name   : TaskName       // Name of the task
    ) : BOOL
```

Input parameter

id

(only short form permitted)

Data type: StructTaskId

TaskId to be checked, e.g. TSI#taskId.

Name

(only short form permitted)

Data type: TaskName

Name of a task, as defined in the execution system.

Return value

Data type: BOOL

The return value indicates whether the TaskId is associated with the specified task:

TRUE: TaskId is associated with the specified task.

FALSE TaskId is not associated with the specified task.

8.3 Functions for runtime measurement of tasks

8.3.1 Functions for runtime measurement of tasks - overview

The functions for runtime measurement are permissible for all tasks. However, the measurement is not supported by the IPOsynchronousTask, the ServosynchronousTask and the ShutdownTask. The functions for runtime measurement of tasks supply the runtime in ms.

The following runtimes can be measured:

- The maximum runtime of the task since the last STOP-RUN transition, see _getMaximalTaskIdRunTime (Page 350) function
- The minimum runtime of the task since the last STOP-RUN transition, see _getMinimalTaskIdRunTime (Page 351) function
- The runtime from the preceding task pass, see _getCurrentTaskIdRunTime (Page 352) function
- The average task runtime from the last ten preceding passes, see _getAverageTaskIdRunTime (Page 353) function

The task is specified for these functions via a unique TaskId.

Note

You can also use the Track Trace to determine the runtime of tasks. For detailed information regarding the Task Trace, see the Task Trace function manual.

Functions for accurate runtime determination

You can use the following system functions to measure the time since the system started with µs precision. Thus, you can measure the time precisely for sections within the application in order, for example, to optimize the application.

- Function _getInternalTimeStamp (Page 354)
- _getTimeDifferenceOfInternalTimeStamps function (Page 355)

See also

Task runtimes (Page 258)

8.3.2 _getMaximalTaskIdRunTime function

Description

The _getMaximalTaskIdRunTime function provides the maximum runtime of the task since the last STOP-RUN transition, including all interruptions by higher-priority tasks. The task is specified by means of a project-wide unique task ID, see description in the section Overview of the task control commands (Page 338).

The following functions do not interrupt the measurement:

- _suspendTaskId (Page 346)
- _disableScheduler (see List Manuals for SIMOTION devices)

The determined runtime is a multiple of the position control cycle clock; for runtimes less than the position controller cycle, T#MIN (= T#0ms) is returned as measured value.

The function is permitted for all tasks. However, the measurement is not supported by the IPOsynchronousTask and the ShutdownTask. Measured value T#MIN (= T#0ms) is returned when called with these tasks.

Declaration

```
_getMaximalTaskIdRunTime (
                { id  : StructTaskId
                }
            ) : TIME
```

Input parameter

id (optional)

Data type: StructTaskId

Default: TaskId of the current task in which the function is called.

TaskId of the task whose runtime should be measured, see description in the section Overview of the task control commands (Page 338).

Return value

Data type:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Measurement is not supported or is not yet finished.
Greater than T#MIN and less than T#MAX	Greatest runtime that has occurred.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId is invalid

8.3.3 _getMinimalTaskIdRunTime function

Description

The _getMinimalTaskIdRunTime function provides the minimum runtime of the task since the last STOP-RUN transition, including all interruptions by higher-priority tasks. The task is specified by means of a project-wide unique task ID, see description in the section Overview of the task control commands (Page 338).

The following functions do not interrupt the measurement:

- _suspendTaskId (Page 346)
- _disableScheduler (see List Manuals for SIMOTION devices)

The determined runtime is a multiple of the position control cycle clock; for runtimes less than the position controller cycle, T#MIN (= T#0ms) is returned as measured value.

The function is permitted for all tasks. However, the measurement is not supported by the IPOsynchronousTask and the ShutdownTask. Measured value T#MIN (= T#0ms) is returned when called with these tasks.

Declaration

```
_getMinimalTaskIdRunTime (
                { id  : StructTaskId
                }
            ) : TIME
```

Input parameter

id (optional)

Data type: StructTaskId

Default: TaskId of the current task in which the function is called.

TaskId of the task whose runtime should be measured, see description in the section Overview of the task control commands (Page 338).

Return value

Data type:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Measurement is not supported or is not yet finished.
Greater than T#MIN and less than T#MAX	Minimum runtime that has occurred.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId is invalid

8.3.4 _getCurrentTaskIdRunTime function

Description

The _getCurrentTaskIdRunTime function provides the runtime from the preceding pass of the task, including all interruptions by higher-priority tasks. The task is specified by means of a project-wide unique task ID, see description in the section Overview of the task control commands (Page 338).

The following functions do not interrupt the measurement:

- _suspendTaskId (Page 346)
- _disableScheduler (see List Manuals for SIMOTION devices)

The determined runtime is a multiple of the position control cycle clock; for runtimes less than the position controller cycle, T#MIN (= T#0ms) is returned as measured value.

The function is permitted for all tasks. However, the measurement is not supported by the IPOsynchronousTask and the ShutdownTask. Measured value T#MIN (= T#0ms) is returned when called with these tasks.

Declaration

```
_getCurrentTaskIdRunTime (
                { id  : StructTaskId
                }
            ) : TIME
```

Input parameter

id (optional)

Data type: StructTaskId

Default: TaskId of the current task in which the function is called.

TaskId of the task whose runtime should be measured, see description in the section Overview of the task control commands (Page 338).

Return value

Data type:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Measurement is not supported or is not yet finished.
Greater than T#MIN and less than T#MAX	Runtime that occurred during the preceding pass.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId is invalid

8.3.5 Function _getAverageTaskIdRunTime

Description

This _getAverageTaskIdRunTime function provides an average runtime of the task from the last ten preceding cycles, including all interruptions by higher-priority tasks. The task is specified by means of a project-wide unique task ID, see description in the section Overview of the task control commands (Page 338).

The following functions do not interrupt the measurement:

- _suspendTask (Page 346)
- _disableScheduler (see List Manuals for SIMOTION devices)

The determined runtime is a multiple of the position control cycle clock; for runtimes less than the position controller cycle, T#MIN (= T#0ms) is returned as measured value.

The function is permitted for all tasks. However, the measurement is not supported by the IPOsynchronousTask and the ShutdownTask. Measured value T#MIN (= T#0ms) is returned when called with these tasks.

Declaration

```
_getAverageTaskIdRunTime (
                { id  : StructTaskId
                }
            ) : TIME
```

Input parameter

id (optional)

Data type	StructTaskId
Default:	TaskID of the task for which runtime is to be measured, as defined in the execution system.

TaskId of the task whose runtime should be measured, see description in the section Overview of the task control commands (Page 338).

Return value

Data type:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Measurement is not supported or is not yet finished.
Greater than T#MIN and less than T#MAX	Average of runtimes that have occurred.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId is invalid

8.3.6 Functions for the precise runtime measurement of tasks

Description

With the following system functions you can measure the time since system start with µs precision. Thus for sections within the application you can measure the time precisely, in order to optimize the application.

You can use the following functions:

- Function _getInternalTimeStamp (Page 354)
- _getTimeDifferenceOfInternalTimeStamps function (Page 355)

8.3.7 Function _getInternalTimeStamp

Description

The _getInternalTimeStamp function supplies a specific internal time stamp as UDINT. You can set a time stamp, e.g. at the start and at the end of a task. You can then use the two time stamps as parameters t1 and t2 of the _getTimeDifferenceOfInternalTimeStamp (Page 355) function.

Note

The function is suitable for the measurement of short times (e.g. performance optimization of algorithms). In the case of extended measurements, there may be differences of the real time and the expired system cycle clocks. These differences can vary for different CPU types.

Declaration

```
_getInternalTimeStamp ()     : UDINT;
```

Input parameter

None

Return value

Data type: UDINT

Internal time stamp.

8.3.8 _getTimeDifferenceOfInternalTimeStamps function

Description

The _getTimeDifferenceOfInternalTimeStamps function supplies a time value from two internal time stamps (UDINT return value). Both time stamps must be generated via the _getInternalTimeStamp (Page 354) function.

Note

When calling the function, pay attention to the order of the parameters t1 and t2 for the internal time stamp. The numeric values of the time stamp are not significant for this.

Declaration

```
_getTimeDifferenceOfInternalTimeStamps (
        t1  : UDINT,  // Internal time stamp 1
        t2  : UDINT   // Internal time stamp 2
    ): UDINT;         // Time difference in µs
```

Input parameter

t1

Data type: UDINT

Internal time stamp 1.

The start time of the interval to be measured, created with _getInternalTimeStamp function.

t2

Data type: UDINT

Internal time stamp 2.

The end time of the interval to be measured, created with _getInternalTimeStamp function.

Return value

Data type: UDINT

Time difference in µs.

Application

With this function you can measure, for example, the runtimes of code sections or communication times in an IPOsynchronous task.

8.4 Functions for message programming (AlarmS)

8.4.1 General information for the message programming

You can send messages configured in SIMOTION SCOUT of the AlarmS type (e.g. error messages) to logged-on display devices and check their status.

In particular, you can:

- Send a message that does not require acknowledgment, see _alarmSId (Page 357) function.
- Send a message that requires acknowledgment, see _alarmSqId (Page 359) function.
- Check the status of a message and its acknowledgment, see _alarmScId (Page 362) function.
- List all pending alarms, see _getPendingAlarms (Page 365) function (as of V4.1).
- Set messages to "outgoing", see _resetAlarmId (Page 366) and _resetAllAlarmId (Page 368) functions (as of V4.1).
- Acknowledge messages, see _acknowledgeAlarmSqId (Page 369) and _acknowledgeAllAlarmSqId (Page 371) functions (as of V4.4).

The message is specified for these functions using a unique AlarmId. For more information about the AlarmId, see the following section (Page 356).

Examples of message programming are included in Programming messages. (Page 471)

See the SIMOTION MCC Programming Manual (Chapters "Incoming message" and "Outgoing message") for programming functions in MCC.

8.4.2 AlarmId

Messages are specified via the associated AlarmId.

You can obtain the AlarmId (data type StructAlarmId) for a configured message name in the following way:

- As variable _alarm.*name*. Explanation of terms:
  - _alarm
    Predefined name space for AlarmId, see Section "Name spaces" in the SIMOTION ST Programming Manual. The identifier for the name space is separated by a period from the following identifier.
  - *name*
    Identifier of the message as configured in SIMOTION SCOUT.
- With the _getAlarmId (*name*) function, see _getAlarmId (Page 364) function.
  name is the identifier of the message as configured in SIMOTION SCOUT.
  This function may only be called in the short form, i.e. with a complete listing of all parameter values, but without specification of the formal parameters.

Note

The _getAlarmId function and variables of the form _alarm.*name* must not be used in libraries.

8.4.3 _alarmSId function

Description

When a level change occurs in the triggering signal (Sig), the function generates a message that does not require acknowledgment and that is sent to all display devices logged on to receive such signals. The message to be generated is specified by means of a project-wide, unique AlarmId, see AlarmId (Page 356).

As option, associated values can be appended:

- Independent of the SIMOTON kernel version:
  One associated value of the general data type ANY_NUM or ANY_BIT
- As of version V4.5 of the SIMOTION kernel:
  Several associated values are possible. They are passed in an ARRAY [0..11] OF BYTE.

A maximum of 40 messages can be processed simultaneously.

Declaration

```
_alarmSId (
         Sig    : BOOL
         Ev_Id  : StructAlarmId
         { sd   : ANY_NUM, ANY_BIT, ARRAY [0..11] OF BYTE
         }
         )  : DWORD
```

Input parameter

Sig

Data type: BOOL

The signal triggering the message is interpreted as follows:

If the signal represents a rising edge – relative to the last call with this AlarmId – an incoming message is generated. An incoming message is also generated if the signal state is TRUE on the first call with this message name.

If the signal represents a falling edge – relative to the last call with this AlarmId – an outgoing message is generated.

Ev_Id

Data type: StructAlarmId

AlarmId of the message to be generated, see AlarmId (Page 356).

sd (optional)

Data type: ANY_NUM, ANY_BIT (independent of the kernel version)
ARRAY [0..11] OF BYTE (as of kernel V4.5)

Default: 0

Associated value:

Associated values must be specified when associated values have been defined during the message configuring in SIMOTION SCOUT.

- Independent of the SIMOTON kernel version:
  One associated value possible, all elementary data types are permissible.
  Variables or previously defined identifiers for constants belonging to one of the permitted data types are possible.
  Values can be entered, but it is not recommended. The format is determined only by the numerical value, which is particularly problematic for floating-point values. If a value is nonetheless specified directly, use the typed literal notation string for constants. A warning is generated if non-typed numerical values are entered.
- As of version V4.5 of the SIMOTION kernel:
  A maximum of 12 associated values of data type BYTE (or SINT, USINT) are possible; they are passed in an ARRAY [0..11] OF BYTE.
  Associated values with data types that comprise several bytes (e.g. WORD, DINT, REAL) are possible; they reduce the number of possible associated values. The user must ensure that their values are stored in the permissible array elements in accordance with the Big Endian byte order. The AnyType_to_BigByteArray (Page 396) function can be used for this purpose.

During compilation, a check cannot be performed with the data types of the associated values configured in the message.

Return value

Data type: DWORD

The return value informs the user about the result of the call.

The specified values can be represented as an OR operation between the ALARMS_ERROR constants and the DSC_SVS_DEVICE_ALARMS_xxx constants.

16#0000	No error For an incoming message, an entry is made in the message list. For an outgoing message, an entry was deleted from the message list.
16#8001	Message name not permitted. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#8002	Loss of messages due to overflow (no more space in the message list). All 40 entries of the message list are occupied. Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED

16#8003	Loss of messages due to overflow (signal not yet sent, signal overflow). Send buffer for notification of the clients is still occupied by the last event. Entry is not made in the message list. Error may also occur if function calls come quickly one after the other with rising and falling edge. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#8004	Double message, message rejected (call with incoming or outgoing message for the second time in succession). Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_CALL
16#8005	No display device registered (message is entered into the list anyway). ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#8006	Message name is already being processed in a lower-priority level. Does not occur for SIMOTION. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
16#8007	No job has been started yet with this message name (first call with Sig = FALSE). Falling edge (outgoing message) arrived without prior rising edge (incoming message) Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#8008	Message name already assigned. Does not occur for SIMOTION. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
16#8009	Internal error ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#8010	Entry was rejected; message acknowledgment memory full. Only for _alarmSqId ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

Example

See Example of message generation (Page 474), Example of messages with several associated values (as of kernel V4.5) (Page 475).

8.4.4 _alarmSqId function

Description

When a level change occurs in the triggering signal (Sig), the function generates a message that requires acknowledgment and that is sent to all display devices logged on to receive such signals. The message to be generated is specified by means of a project-wide, unique AlarmId, see AlarmId (Page 356).

As option, associated values can be appended.

- Independent of the SIMOTON kernel version:
  One associated value of the general data type ANY_NUM or >ANY_BIT
- As of version V4.5 of the SIMOTION kernel:
  Several associated values are possible. They are passed in an ARRAY [0..11] OF BYTE.

A maximum of 40 messages can be processed simultaneously.

Declaration

```
_alarmSqId (
         Sig    : BOOL
         Ev_Id  : StructAlarmId
         { sd   : ANY_NUM, ANY_BIT, ARRAY [0..11] OF BYTE
         }
         )  : DWORD
```

Input parameter

Sig

Data type: BOOL

The signal triggering the message is interpreted as follows:

If the signal represents a rising edge – relative to the last call with this message name – an incoming message is generated. An incoming message is also generated if the signal state is TRUE on the first call with this message name.

If the signal represents a falling edge – relative to the last call with this message name – an outgoing message is generated.

If no signal is generated with falling edge, an alarm of type SQ remains in the message buffer, even if it has been acknowledged using _acknowledgeAllAlarmSqId.

Ev_Id

Data type: StructAlarmId

AlarmId of the message to be generated, see AlarmId (Page 356).

sd (optional)

Data type:	ANY_NUM, ANY_BIT (independent of the kernel version) ARRAY [0..11] OF BYTE (as of kernel V4.5)
Default:	0

Associated value

Associated values must be specified when associated values have been defined during message configuring in SIMOTION SCOUT.

- Independent of the SIMOTON kernel version:
  One associated value possible, all elementary data types are permissible.
  Variables or previously defined identifiers for constants belonging to one of the permitted data types are possible.
  Values can be entered, but it is not recommended. The format is determined only by the numerical value, which is particularly problematic for floating-point values. If a value is nonetheless specified directly, use the typed literal notation string for constants. A warning is generated if non-typed numerical values are entered.
- As of version V4.5 of the SIMOTION kernel:
  A maximum of 12 associated values of BYTE (or SINT, USINT) data type is possible; they are passed in an ARRAY [0..11] OF BYTE.
  Associated values with data types that comprise several bytes (e.g. WORD, DINT, REAL) are possible; they reduce the number of possible associated values. The user must ensure that their values are stored in the permissible array elements in accordance with the Big Endian byte order. The AnyType_to_BigByteArray (Page 396) function can be used for this purpose.

During compilation, a check cannot be performed with the data type of the associated value configured in the message.

Return value

Data type: DWORD

The return value informs the user about the result of the call.

The specified values can be represented as an OR operation between the ALARMS_ERROR constants and the DSC_SVS_DEVICE_ALARMS_xxx constants.

16#0000	No error For an incoming message, an entry is made in the message list. For an outgoing message, an entry was deleted from the message list.
16#8001	Message name not permitted. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#8002	Loss of messages due to overflow (no more space in the message list). All 40 entries of the message list are occupied. Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED

16#8003	Loss of messages due to overflow (signal not yet sent, signal overflow). Send buffer for notification of the clients is still occupied by the last event. Entry is not made in the message list. Error may also occur if function calls come quickly one after the other with rising and falling edge. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#8004	Double message, message rejected (call with incoming or outgoing message for the second time in succession). Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_CALL
16#8005	No display device registered (message is entered into the list anyway). ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#8006	Message name is already being processed in a lower-priority level. Does not occur for SIMOTION. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
16#8007	No job has been started yet with this message name (first call with Sig = FALSE). Falling edge (outgoing message) arrived without prior rising edge (incoming message) Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#8008	Message name already assigned. Does not occur for SIMOTION. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
16#8009	Internal error ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#8010	Entry was rejected; message acknowledgment memory full. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

Example

See Example of message generation (Page 474), Example of messages with several associated values (as of kernel V4.5) (Page 475).

8.4.5 _alarmScId function

Description

The function displays the status of a message and its acknowledgment status. The message is specified by means of a project-wide, unique AlarmId, see AlarmId (Page 356).

Alarms that do not have to be acknowledged are always treated as acknowledged in "incoming" status.

Declaration

```
_alarmScId (
         Ev_Id    : StructAlarmId
         )    : DWORD
```

Input parameter

Ev_Id

Data type: StructAlarmId

AlarmId of the relevant message, see AlarmId (Page 356).

Return value

Data type: DWORD

The return value informs the user about the result of the call and the status of a message. The constant values and symbolic constants below can be used with equal priority.

First check for errors:

16#8000 Filter for errors
ALARMS_ERROR

16#8001 Message name not permitted.
ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID

Second check for message status:

16#0000 Outgoing message, not acknowledged (a).

16#0001 Incoming message, not acknowledged (b).
ALARMS_STATE

16#0010 There is no message for this message name.
Three options:
1. Message never triggered
2. Message triggered via _alarmSId, but also sent
3. Message triggered via _alarmSqId, sent and acknowledged at the display device

16#0101 Incoming message, acknowledged (c).
ALARMS_STATE OR ALARMS_QSTATE

Messages (a) ... (b) imply _alarmSqId.

Message (c) implies _alarmSqId and _alarmSId.

Example

See Checking the error number and status of a message (filtering return values) (Page 477).

8.4.6 _getAlarmId function

Description

The _getAlarmId function generates the AlarmId (Page 356) of the message from a configured, application-specific message name. This AlarmId can be assigned to a variable of data type StructAlarmId and used as an input parameter in the following functions:

- _alarmSId (Page 357) function
- _alarmSqId (Page 359) function
- _alarmScId (Page 362) function
- _resetAlarmId (Page 366) function
- _acknowledgeAlarmSqId (Page 369) function

Note

This function must not be used in libraries.

This function may only be called in the short form, i.e. with a complete listing of all parameter values, but without specification of the formal parameters.

Declaration

```
_getAlarmId (  // Only the short form is permitted
        Al_Name : Alarm_Name // Name of the message
        ) : StructAlarmId
```

Input parameter

Al_Name

This is a message name configured specifically for the application that is created when the message is configured in SIMOTION SCOUT.

Return value

Data type: StructAlarmId

The return value contains the AlarmId of the configured message, see AlarmId (Page 356).

8.4.7 Function _getPendingAlarms

Description

The _getPendingAlarms provides information on the pending alarms. It provides:

- The number of entries in the message list (maximum of 40 entries).
- For each entry in the message list:
  - The AlarmId (Page 356) of the message
  - The message type:
    Message that does not require acknowledgement generated with the _alarmSId (Page 357), or
    Message that requires acknowledgement generated with the _alarmSqId (Page 359).
  - The status of the message: "Incoming" or "Outgoing".

The function is available for SIMOTION devices as of version V4.1.

Declaration

```
_getPendingAlarms ()  :StructRetGetPendingAlarms
```

Input parameter

None

Return value

Data type: StructRetGetPendingAlarmState

The return value is a structure of data type *StructRetGetPendingAlarmState*.

It comprises the following:

- A numberOfPendingAlarms component: UINT
  This provides information on the number of entries in the message list.
- An alarm component: ARRAY [0..39] OF StructPendingAlarmState
  This provides information about the state of the individual messages in the list (maximum of 40 entries).
  The state of the individual messages is a structure of data type *StructPendingAlarmState*. It comprises the following:
  - An Id component: StructAlarmId
    This provides information about the project-wide, unique AlarmId (Page 356) of the message
  - A _type component: EnumAlarmIdType
    This provides information about the message type (requires acknowledgement or not)
  - A state component: EnumAlarmIdState
    This provides information about the message state (incoming or outgoing)

Data types

```
TYPE
  EnumAlarmIdType  : (
    ALARM_S ,   // Message that does not require acknowledgement
    ALARM_SQ ); // Message that requires acknowledgement
  EnumAlarmIdState : (
    INCOMING_ALARM ,   // Incoming message
    OUTGOING_ALARM );  // Outgoing message
  StructPendingAlarmState    : STRUCT
    Id     : StructAlarmId;
    _type  : EnumAlarmIdType;
    state  : EnumAlarmIdState;
  END_STRUCT;
  StructRetGetPendingAlarmState : STRUCT
    numberOfPendingAlarms  : UINT;
    alarm  : ARRAY [0..39] OF StructPendingAlarmState;
  END_STRUCT;
END_TYPE
```

8.4.8 _resetAlarmId function

Description

The _resetAlarmId function sets a message to "outgoing". The message is specified by means of a project-wide, unique AlarmId, see AlarmId (Page 356).

If this is a message that requires acknowledgment that has been generated with the _alarmSqId (Page 359) function, the following applies:

- A message that has not been acknowledged must also be acknowledged, e.g.
  - Via the HMI or SIMOTION SCOUT.
  - With the _acknowledgeAlarmSqId (Page 369) or _acknowledgeAllAlarmSqId (Page 371) functions.
- An already acknowledged message is deleted.

A message that does not require acknowledgment that has been generated with the _alarmSId (Page 357) function is deleted:

The function is available for SIMOTION devices as of version V4.1.

Declaration

```
_resetAlarmId (
        Id  : StructAlarmId
        ) : UDINT
```

Input parameter

Id

Data type: StructAlarmId

AlarmId of the message that is to be set to "outgoing", see AlarmId (Page 356).

Return value

Data type UDINT

The return value informs the user about the result of the call.

16#0000_0000	No error Message could be acknowledged without errors.
16#0000_8001	Message name not permitted. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#0000_8002	Loss of messages due to overflow (no more space in the message list). All 40 entries of the message list are occupied. Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
16#0000_8003	Loss of messages due to overflow (signal not yet sent, signal overflow). Send buffer for notification of the clients is still occupied by the last event. Entry is not made in the message list. Error may also occur if function calls come quickly one after the other with rising and falling edge. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#0000_8005	No display unit registered (message is entered into the list anyway). ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#0000_8007	No job has been started yet with this message name (first call with Sig = FALSE). Falling edge (outgoing message) arrived without prior rising edge (incoming message) Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#0000_8009	Internal error. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#0000_8010	Entry was rejected; message acknowledgment memory full. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

8.4.9 _resetAllAlarmId function

Description

The _resetAllAlarmId function sets all messages to "outgoing".

For messages that require acknowledgment that have been generated with the _alarmSqId (Page 359) function, the following applies:

- Messages that have not been acknowledged must also be acknowledged, e.g.
  - Via the HMI or SIMOTION SCOUT.
  - With the _acknowledgeAlarmSqId (Page 369) or _acknowledgeAllAlarmSqId (Page 371) functions.
- Already acknowledged messages are deleted.

Messages that do not require acknowledgment that have been generated with the _alarmSId (Page 357) function are deleted:

The function is available for SIMOTION devices as of version V4.1.

Declaration

```
_resetAllAlarmId  () : UDINT
```

Input parameter

None

Return value

Data type	UDINT
The return value informs the user about the result of the call.	
16#0000_0000	No error Message could be acknowledged without errors.
16#0000_8001	Message name not permitted. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#0000_8002	Loss of messages due to overflow (no more space in the message list). All 40 entries of the message list are occupied. Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED

16#0000_8003	Loss of messages due to overflow (signal not yet sent, signal overflow). Send buffer for notification of the clients is still occupied by the last event. Entry is not made in the message list. Error may also occur if function calls come quickly one after the other with rising and falling edge. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#00008005	No display unit registered (message is entered into the list anyway). ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#0000_8007	No job has been started yet with this message name (first call with Sig = FALSE). Falling edge (outgoing message) arrived without prior rising edge (incoming message) Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#0000_8009	Internal error. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#0000_8010	Entry was rejected; message acknowledgment memory full. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

8.4.10 _acknowledgeAlarmSqId function

Description

The _acknowledgeAlarmSqId function acknowledges a message that requires acknowledgment that has been generated with the _alarmSqId (Page 359) function. The message is specified by means of a project-wide, unique AlarmId, see AlarmId (Page 356).

This enables the user, for example, to acknowledge a message via the signal of an I/O input and the call of the function.

The function is available for SIMOTION devices as of version V4.4.

Declaration

```
_acknowledgeAlarmSqId (
        Id  : StructAlarmId
        ) : DWORD
```

Input parameter

Id

Data type: StructAlarmId

AlarmId of the message that is to be acknowledged, see AlarmId (Page 356).

Return value

Data type	DWORD
The return value informs the user about the result of the call.	
16#0000_0000	No error Message could be acknowledged without errors.
16#0000_8001	Message name not permitted. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#0000_8002	Loss of messages due to overflow (no more space in the message list). All 40 entries of the message list are occupied. Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
16#0000_8003	Loss of messages due to overflow (signal not yet sent, signal overflow). Send buffer for notification of the clients is still occupied by the last event. Entry is not made in the message list. Error may also occur if function calls come quickly one after the other with rising and falling edge. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#0000_8005	No display unit registered (message is entered into the list anyway). ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#0000_8007	No job has been started yet with this message name (first call with Sig = FALSE). Falling edge (outgoing message) arrived without prior rising edge (incoming message) Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#0000_8009	Internal error. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#0000_8010	Entry was rejected; message acknowledgment memory full. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

8.4.11 _acknowledgeAllAlarmSqId function

Description

The _acknowledgeAllAlarmSqId function acknowledges all messages that require acknowledgment that have been generated with the _alarmSqId (Page 359) function.

This enables the user, for example, to acknowledge all messages via the signal of an I/O input and the call of the function.

The function is available for SIMOTION devices as of version V4.4.

Declaration

```
_acknowledgeAllAlarmSqId  () : DWORD
```

Input parameter

None

Return value

Data type	DWORD
The return value informs the user about the result of the call.	
16#0000_0000	No error Message could be acknowledged without errors.
16#0000_8001	Message name not permitted. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#0000_8002	Loss of messages due to overflow (no more space in the message list). All 40 entries of the message list are occupied. Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
16#0000_8003	Loss of messages due to overflow (signal not yet sent, signal overflow). Send buffer for notification of the clients is still occupied by the last event. Entry is not made in the message list. Error may also occur if function calls come quickly one after the other with rising and falling edge. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#0000_8005	No display unit registered (message is entered into the list anyway). ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED

16#0000_8007	No job has been started yet with this message name (first call with Sig = FALSE). Falling edge (outgoing message) arrived without prior rising edge (incoming message) Entry is not made in the message list. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#0000_8009	Internal error. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#0000_8010	Entry was rejected; message acknowledgment memory full. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

Programming of general standard functions 9

9.1 Programming of general standard functions - overview

SIMOTION provides a series of standard functions you can call in your sources to solve common tasks.

Note

Some of the following standard functions can only be called in short form, i.e. with a complete list of all parameter values, but without specifying the formal parameters:

- Result := function name (1st parameter value, 2nd parameter value)
  instead of
- Result := function name (formal parameter1 := 1st parameter value, etc.)

This is noted in the description of each function.

For information about the general data types (e.g. ANY_INT, ANY_BIT), see **Elementary data types**, General data types table in the ST programming manual.

Using the command library

The command library is a tab in the project navigator. It contains the available system functions, system function blocks, and operators.

You can use a drag-and-drop operation to move these elements from the command library:

- To the window of the ST editor
- To a network of the LAD/FBD editor
- To the system function call command of the MCC editor

Comparison of the system functions for SIMOTION and SIMATIC

You can find a comparison of the SIMATIC S7 and SIMOTION system functions on the Utilities & Applications CD under FAQs.

9.2 Numeric standard functions

9.2.1 Special features of a numeric function

Every numeric standard function has an input parameter. The result is always the return value. The general numeric, the logarithmic and the trigonometric standard functions each specify a group of numeric standard functions through the function name and the data type.

9.2.2 General numeric standard functions

General numeric standard functions are used for:

- Calculation of the absolute value of a variable
- Calculation of the square root of a variable
- Truncating a variable to its integer part

Table 9-1 General numeric standard functions

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
ABS	in	ANY_NUM	ANY_NUM[1]	Absolute value
SQRT	in	ANY_REAL	ANY_REAL[1]	Square root
TRUNC	in	ANY_REAL	ANY_INT	Truncation of value to integer part (0 direction)

[1] Data type of input parameter *in*.

The following table shows possible general numeric standard function calls and their results:

Table 9-2 General numeric standard function calls

Call	Result
`Result := ABS (-5);` `Result := ABS (in := -5);`	`5`
`Result := SQRT (81.0);` `Result := SQRT (in := 81.0);`	`9.0`
`Result := TRUNC (-3.141_59);` `Result := TRUNC (in := -3.141_59);`	`-3`

9.2.3 Logarithmic standard functions

Logarithmic standard functions are the functions for the calculation of an exponential value or a logarithm.

Table 9-3 Logarithmic standard functions

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
EXP	in	ANY_REAL	ANY_REAL[1]	e^x (e function)
EXPD	in	ANY_REAL	ANY_REAL[1]	10^x
EXPT	in1	ANY_REAL[2]	ANY_REAL[3]	Exponentiation (see also Operator ** in **Arithmetic expressions** in the ST programming manual)
	in2	ANY_REAL		

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
LN	in	ANY_REAL	ANY_REAL[1]	Natural logarithm
LOG	in	ANY_REAL	ANY_REAL[1]	Common logarithm

1 Data type of input parameter *in*.

2 The input parameter *in1* must be greater than zero.
Exceptions as of SIMOTION Kernel V4.1:
– If *in2* is an integer, *in1* can also be less than zero.
– If *in2* is positive, *in1* can also be equal to zero.
The following applies up to SIMOTION Kernel V4.0: If *in1* is equal to zero, an error message can be caught with ExecutionFaultTask.

3 Data type of input parameter *in1*.

The following table shows possible logarithmic standard function calls and their results:

Table 9-4 Logarithmic standard function calls

Call	Result
`Result := EXP (4.1);` `Result := EXP (in := 4.1);`	`60.3403 ...`
`Result := EXPD (3.0);` `Result := EXPD (in := 3.0);`	`1_000.0`

`Result := EXPT (2.0,4.0);`
`Result := EXPT (in1 := 2.0, in2 := 4.0);`
`Result := 2.0 ** 4.0;`
	`16.0`

`Result := LN (2.718_281);` `Result := LN (in := 2.718_281);`	`1.0`
`Result := LOG (245);` `Result := LOG (in := 245);`	`2.389_166`

9.2.4 Trigonometric standard functions

The trigonometric standard functions shown expect and calculate variables of angles in radians.

Table 9-5 Trigonometric standard functions

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
ACOS	in	ANY_REAL	ANY_REAL[1 2]	Arc cosine (main value)
ASIN	in	ANY_REAL	ANY_REAL[1 2]	Arc sine (main value)
ATAN	in	ANY_REAL	ANY_REAL[1 2]	Arc tangent (main value)
COS	in	ANY_REAL[2]	ANY_REAL[1]	Cosine (radian measure input)
SIN	in	ANY_REAL[2]	ANY_REAL[1]	Sine (radian measure input)
TAN	in	ANY_REAL[2]	ANY_REAL[1]	Tangent (radian measure input)

1 Data type of input parameter *in*.

2 Radian angle

The following table shows possible trigonometric standard function calls and their results:

Table 9-6 Trigonometric standard function calls

Call	Result

`PI := 3.141592; //PI is a variable!`
`Result := SIN (PI / 6);`
`Result := SIN (in := PI / 6);`
	`0.5`

`Result := ACOS (0.5);` `Result := ACOS (in := 0.5);`	`1.047_197 //equals PI/3`

9.2.5 Bit string standard functions

Each bit string standard function has two input parameters:

- in (data type ANY_BIT):

Bit string to be manipulated by bit shift operations

- n (data type USINT):
  - Number of places to be rotated for ROL and ROR
  - Number of places to be shifted for SHL and SHR

The result is always the return value. The following table shows the function names and the data types of the two input parameters and of the return value.

Table 9-7 Bit string standard functions

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
ROL	in n	ANY_BIT USINT	ANY_BIT[1]	The bit string in parameter *in* is **left-rotated** through the number of places specified by the content of parameter *n*.
ROR	in n	ANY_BIT USINT	ANY_BIT[1]	The bit string in parameter *in* is **right-rotated** through the number of places specified by the content of parameter *n*.
SHL	in n	ANY_BIT USINT	ANY_BIT[1]	The bit string contained in parameter *in* is **left-shifted** and replaced by 0, as specified by the content of parameter *n* [2].

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
SHR	in n	ANY_BIT USINT	ANY_BIT[1]	The bit string contained in parameter *in* is **right-shifted** and replaced by 0, as specified by the content of parameter *n* [2].

[1] Data type of input parameter *in*
[2] Only the five least-significant bits of the parameter in are evaluated, for example:
SHL (16#FFFF_FFFF,32) = 16#FFFF_FFFF
SHR (16#FFFF_FFFF,32) = 16#FFFF_FFFF

1 Data type of input parameter *in*.

2 Only the five least-significant bits of the parameter in are evaluated, for example:
SHL (16#FFFF_FFFF,32) = 16#FFFF_FFFF
SHR (16#FFFF_FFFF,32) = 16#FFFF_FFFF

Note

If numerical values are used as input parameters, the smallest possible data type is assumed (for example, BOOL if 1, BYTE if 2).

The following table shows possible bit string standard function calls and their results.

Table 9-8 Examples of bit string standard function calls

Call	Result
`Result := ROL (2#1101_0011, 5);` `// 1st parameter corresponds to 211 decimal`	`2#0111_1010` `(= 122 decimal)`
`Result := ROR (2#1101_0011, 2);` `// 1st parameter corresponds to 211 decimal`	`2#1111_0100` `(= 244 decimal)`
`Result := SHL (2#1101_0011, 3);` `// 1st parameter corresponds to 211 decimal`	`2#1001_1000` `(= 152 decimal)`
`Result := SHR (2#1101_0011, 2);` `// 1st parameter corresponds to 211 decimal`	`2#0011_0100` `(= 52 decimal)`
`Result := SHL (1, 3);` `// 1st parameter data type BOOL`	`2#0000_0000` `(= 0 decimal)`
`Result := SHL (2, 3);` `// 1st parameter data type BYTE`	`2#0001_0000` `(= 16 decimal)`

9.3 Access to bits in bit strings

9.3.1 _getBit function

Description

This function reads the bit string variable and returns the value of the specified bit.

Note

The access to bits of an I/O variable or system variable can be interrupted by other tasks. There is therefore no guarantee of consistency.

Access to I/O variables of the BOOL data type is consistent.

Declaration

```
FUNCTION _getBit (
        in  : ANY_BIT,  // Bit string variable
        n   : USINT     // Number of the bit
    ) : BOOL
```

Input parameter

in

Data type ANY_BIT

Bit string variable

n

Data type USINT

Number of the bit for which the value is to be output

Permissible values:		
	[0..7]	For BYTE
	[0..15]	For WORD
	[0..31]	For DWORD

When specifying impermissible values, the function supplies the return value FALSE.

Return value

Data type: BOOL

Value of the bit

Example

```
myBit := _getBit (
                in  := myBitString,
                n   := 5);
```

The user variable *myBit* contains bit 5 of the user variable *myBitString*.

Direct bit access

You can also access an individual bit of a bit string variable by specifying the number of the bit as constant separated by a point behind the variable. The number of the bit must be specified as a constant of the ANY_INT data type, see also SIMOTION ST Programming and Operating Manual.

To be able to use this option, you must activate the "Permit language extensions" compiler option (Page 322), see also SIMOTION ST Programming and Operating Manual.

```
FUNCTION f : VOID
    VAR CONSTANT
        BIT_7 : INT := 7;
    END_VAR
    VAR
        dw : DWORD;
        b  : BOOL;
    END_VAR
    b := dw.BIT_7; // Access to bit number 7
    b := dw.3;     // Access to bit number 3
//  b := dw.33;    // Compilation error:
//  Bit offset not permitted for the data type!
END_FUNCTION
```

Note

The access to bits of an I/O variable or system variable can be interrupted by other tasks. There is therefore no guarantee of consistency.

9.3.2 _setBit function

Description

The function reads the specified bit string variable and returns its value. The specified bit is set to the prespecified Boolean value (TRUE or FALSE).

To change a bit value to a variable with this function, reassign the return value to this variable.

Example:
`myVar := _setBit (in := myVar, n := 2, value := TRUE);`

```
// Bit 2 is set to the value 1 in the myVar variable.
```

Note

A function call is generally executed in 3 steps:

- Read the input variables
- Function execution
- Result assignment

The execution sequence can be interrupted by other tasks. This also applies to the _setBit function.

When you use the function to modify bits of a variable, limit all function calls for the same variable to one task, e.g. BackgroundTask, IpoSynchronousTask, MotionTask_x.

The use for I/O variables (address list) is not recommended because interruptions by other accesses to the I/O variables are possible, e.g. access to lower-level BOOL variable declared in the address list, update of the process image. This can result in undesirable data corruption.

Declaration

```
FUNCTION _setBit (
        in      : ANY_BIT,    // Bit string variable
        n       : USINT,      // Number of the bit
      { value   : BOOL        // Value of the bit
        }
    ) : ANY_BIT
```

Input parameter

in

Data type: ANY_BIT

Bit string variable

n

Data type: USINT

Number of the bit for which the value is to be set.

Permissible values:		
	[0..7]	For BYTE
	[0..15]	For WORD
	[0..31]	For DWORD

If illegal values are specified (without an additional message), the unchanged value of the bit string variable is returned.

value (optional)

Default setting: TRUE

Value assigned to the bit to be set

Return value

Data type: ANY_BIT
Data type of input parameter *in*.
Value of the bit string variable with modified bit.

Example

```
myBitString := _setBit (
                    in    := myBitString,
                    n     := 5,
                    value := FALSE);
```

Bit 5 of the *myBitString* user variable is set to FALSE (logic 0).

Direct bit access

You can also access an individual bit of a bit string variable by specifying the number of the bit as constant separated by a point behind the variable. The number of the bit must be specified as a constant of the ANY_INT data type, see also SIMOTION ST Programming and Operating Manual.

To use this option, you must activate the "Permit language extensions" compiler option (Page 322), see also SIMOTION ST Programming and Operating Manual.

```
FUNCTION f : VOID
    VAR CONSTANT
        BIT_7 : INT := 7;
    END_VAR
    VAR
        dw : DWORD;
        b  : BOOL;
    END_VAR
    b  := 1;
    dw.BIT_7 := b; // Write to bit number 7
    dw.3     := b; // Write to bit number 3
//  dw.33    := b; // Compilation error:
//  Bit offset not permitted for the data type!
END_FUNCTION
```

Note

When you set or reset multiple bits of a variable, limit all function calls for the same variable to one task, e.g. BackgroundTask, IpoSynchronousTask, MotionTask_x.

The use for I/O variables (address list) is not recommended because interruptions by other accesses to the I/O variables are possible, e.g. access to lower-level BOOL variable declared in the address list, update of the process image. This can result in undesirable data corruption.

9.3.3 _toggleBit function

Description

This function reads the specified bit string variable and returns its value, whereby the specified bit is inverted.

Change the specified bit in the variable by assigning the return value to the variable again.

Note

A function call is generally executed in 3 steps:

- Read the input variables
- Function execution
- Result assignment

The execution sequence can be interrupted by other tasks. This also applies to the _setBit function.

When you use the function to modify bits of a variable, limit all function calls for the same variable to one task, e.g. BackgroundTask, IpoSynchronousTask, MotionTask_x.

The use for I/O variables (address list) is not recommended because interruptions by other accesses to the I/O variables are possible, e.g. access to lower-level BOOL variable declared in the address list, update of the process image. This can result in undesirable data corruption.

Declaration

```
FUNCTION _toggleBit (
        in  : ANY_BIT,  // Bit string variable
        n   : USINT,    // Number of the bit
    ) : ANY_BIT
```

Input parameter

in

Data type: ANY_BIT

Bit string variable

n

Data type: USINT

Number of the bit whose value is to be inverted (set from TRUE to FALSE or from FALSE to TRUE).

Permissible values:		
	[0..7]	For BYTE
	[0..15]	For WORD
	[0..31]	For DWORD

If illegal values are specified (without an additional message), the unchanged value of the bit string variable is returned.

Return value

Data type: ANY_BIT
Data type of input parameter *in*.
Value of the bit string variable with inverted bit

Example

```
myBitString := _toggleBit (
                    in  := myBitString,
                    n   := 5);
```

Bit 5 of the *myBitString* user variable is inverted.

Direct bit access

You can also access an individual bit of a bit string variable by specifying the number of the bit as constant separated by a point behind the variable. The number of the bit must be specified as a constant of the ANY_INT data type, see also SIMOTION ST Programming and Operating Manual.

To be able to use this option, you must activate the "Permit language extensions" compiler option (Page 322), see also SIMOTION ST Programming and Operating Manual.

```
FUNCTION f : VOID
    VAR CONSTANT
        BIT_7 : INT := 7;
    END_VAR
    VAR
        dw : DWORD;
        b  : BOOL;
    END_VAR
    b  := 1;
    dw.BIT_7 := NOT dw.BIT_7; // Write to bit number 7
    dw.3     := NOT dw.3;     // Write to bit number 3
//  dw.33    := NOT dw.33;    // Compilation error:
//  Bit offset not permitted for the data type
END_FUNCTION
```

Note

When you set or reset multiple bits of a variable, limit all function calls for the same variable to one task, e.g. BackgroundTask, IpoSynchronousTask, MotionTask_x.

The use for I/O variables (address list) is not recommended because interruptions by other accesses to the I/O variables are possible, e.g. access to lower-level BOOL variable declared in the address list, update of the process image. This can result in undesirable data corruption.

9.4 Bit operations on numeric data types

The following functions enable bit operations on numeric data types. Each bit of the return value is generated from the corresponding bits of the input parameters.

Table 9-9 Bit operators on numeric data types

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
_NOT	in	ANY_INT	ANY_INT[1]	Bit-by-bit negation
_AND	in1 in2	ANY_INT[32] ANY_INT[32]	ANY_INT[43]	Bit-by-bit conjunction (AND operation): A bit of the return value is only 1 when all the corresponding bits of the input parameters are 1, otherwise 0.
_OR	in1 in2	ANY_INT[32] ANY_INT[32]	ANY_INT[4]	Bit-by-bit disjunction (OR operation): A bit of the return value is only 1 when at least one of the corresponding bits of the input parameters is 1, otherwise 0.
_XOR	in1 in2	ANY_INT[32] ANY_INT[32]	ANY_INT[43]	Bit-by-bit exclusive disjunction (exclusive OR operation): A bit of the return value is only 1 when exactly one of the corresponding bits of the input parameters is 1, otherwise 0.

1 Data type of the input parameter.

2 It must be possible to convert the data types of in1 and in2 implicitly to a common general ANY_INT data type.

3 Smallest common data type to which the input parameters can be converted implicitly.

9.5 String processing

9.5.1 Functions for the string editing

The following functions enable the processing of variables of data type STRING.

Table 9-10 Functions for the string processing

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
CONCAT	in1 in2	STRING[254] STRING[254]	STRING[254]	Attaches string in2 to string in1[1]. Error flags: TSI#ERRNO := 1, if LEN (in1) + LEN (in2) > 254
DELETE	in l p	STRING[254] INT INT	STRING[254]	Deletes l character(s) from string in, starting at position p[2 4 6]. Error flags: TSI#ERRNO := 2, if l < 0, p < 1, p > LEN (in)

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
FIND	in1 in2	STRING[254] STRING[254]	INT	Returns the position at which string in2 begins in string in1. Result is 0 if string in2 is not contained in string in1. Error flags: None
INSERT	in1 in2 p	STRING[254] STRING[254] INT	STRING[254]	Inserts string in2 in string in1, starting at position p[1 2 5 7]. Error flags: TSI#ERRNO := 2, if p < 0, p > LEN (in1) TSI#ERRNO := 1, if LEN (in1) + LEN (in2) > 254
LEFT	in l	STRING[254] INT	STRING[254]	Returns the first l characters in string in[2 3]. Error flags: TSI#ERRNO := 2, if l < 0
LEN	in	STRING[254]	INT	Returns the number of characters in string in. Error flags: None
MID	in l p	STRING[254] INT INT	STRING[254]	Returns l characters from string in, starting at position p[2 3]. Error flags: TSI#ERRNO := 2, if l < 0, p < 1, p > LEN (in)
REPLACE	in1 in2 l p	STRING[254] STRING[254] INT INT	STRING[254]	Replaces l characters from string in1 with string in2, starting at position p[2 4 7]. Error flags: TSI#ERRNO := 2, if l < 0, p < 1, p > LEN (in1) TSI#ERRNO := 1, if LEN (in1) + LEN (in2) - l > 254
RIGHT	in l	STRING[254] INT	STRING[254]	Returns the last l characters in string in[2 3]. Error flags: TSI#ERRNO := 2, if l < 0

1 If LEN (in1) + LEN (in2) > 254: String is truncated.

2 If l < 0 or p < 0: Empty string is returned.

3 If l = 0 or p = 0: Empty string is returned.

4 If l = 0 or p = 0: String in or in1 remains unchanged.

5 If p = 0: String in1 is attached to string in2.

6 If p > LEN(in) : String in remains unchanged.

7 If p > LEN(in1): String in2 is attached to string in1.

Table 9-11 Examples for calls of the functions for the string processing

Call	Result
`A := CONCAT (in1 := 'ASTRING', in2 := '123');`	`'ASTRING123'.`

`A := DELETE (in1 := 'ASTRING', l := 2, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := 2, p := 0);`
`A := DELETE (in1 := 'ASTRING', l := 2, p := 8);`
`A := DELETE (in1 := 'ASTRING', l := 0, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := 10, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := -1, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := 2, p := -1);`
	`'ASTNG'.`
	`'ASTRING'.`
	`'ASTRING'.`
	`'ASTRING'.`
	`'AST'.`
	`''.`
	`''.`

`B := FIND (in1 := 'ASTRING', in2 := 'RI');` `B := FIND (in1 := 'ASTRING', in2 := 'RB');`	`4.` `0.`

`A := INSERT (in1 := 'ASTRING', in2 := '123', p := 1);`
`A := INSERT (in1 := 'ASTRING', in2 := '123', p := 0);`
`A := INSERT (in1 := 'ASTRING', in2 := '123', p := 10);`
`A := INSERT (in1 := 'ASTRING', in2 := '123', p :=-1);`
	`'A123STRING'.`
	`'123ASTRING'.`
	`'ASTRING123'.`
	`''.`

`A := LEFT (in := 'ASTRING', l := 3);`
`A := LEFT (in := 'ASTRING', l := 10);`
`A := LEFT (in := 'ASTRING', l := -1);`
	`'AST'.`
	`'ASTRING'.`
	`''.`

`B := LEN (in := 'ASTRING');`	`7.`

`A := MID (in := 'ASTRING', l :=3, p :=2 );`
`A := MID (in := 'ASTRING', l :=3, p :=6 );`
`A := MID (in := 'ASTRING', l :=3, p :=8 );`
`A := MID (in := 'ASTRING', l :=3, p :=0 );`
	`'STR'.`
	`'NG'.`
	`''.`
	`''.`

`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 2);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 1);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 0, p := 2);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 0);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 2, p := 10);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 5);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := -1);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := -1, p : =2);`
	`'A123NG'.`
	`'123ING'.`
	`'ASTRING'.`
	`'ASTRING'.`
	`'ASTRING123'.`
	`'ASTRI123'.`
	`''.`
	`''.`

`A := RIGHT (in := 'ASTRING', l := 3);`
`A := RIGHT (in := 'ASTRING', l := 10);`
`A := RIGHT (in := 'ASTRING', l := -1);`
	`'ING'.`
	`'ASTRING'.`
	`''.`

For information about conversion functions for STRINGs, see Functions for the conversion of numeric data types to STRING data type (Page 394).

9.5.2 Error analysis during the string editing

Description

Errors which occur during string functions are stored separately for each task in the Taskstartinfo. It is therefore implemented in the task context and can then be subsequently queried accordingly in the same task, e.g. BackgroundTask.

```
Variable:TSI#ERRNO : DINT
```

Value 0 indicates free of errors. For the string functions, the errors in P (position in the string) and L (number of characters) are stored separately from the violation of the maximum string length with different values.

- Value 0: for free of errors
- Value 1: for violation of the maximum string length
- Value 2: for p or l outside the value range

Note

TSI#ERRNO cannot be used to examine the string length (applies to DINT_TO_STRING, UDINT_TO_STRING, REAL_TO_STRING, and LREAL_TO_STRING).

In this case, the 0 error code will be set in TSI#ERRNO (string length exceeded).

You must, therefore, ensure that the string you enter is long enough or should check the length in the user program prior to conversion.

Resetting of the TSI#ERRNO value:

- The error remains in Taskstartinfo until you explicitly reset the content of TSI#ERRNO.
- The value of TSI#ERRNO is only rewritten in the case of an error when executing a new string function.
  A correctly executed string function does not overwrite any error ID present in TSI#ERRNO.
- The error ID is reset when the task is restarted.
  Cyclic execution of a task (e.g. IpoSynchronousTask, BackgroundTask) does not overwrite the error ID. The error ID is retained, as is the content of local variables.

Table 9-12 Example

```
VAR
    A : STRING [254];
END_VAR

A := DELETE (in := 'ASTRING', l:= -1, p := 4);
// Result is an empty string
IF (TSI#ERRNO = 2)THEN  // Error during string processing
    // l < 0, p < 1, p > LEN (in)
   A := 'ERROR';
END_IF;
```

9.6 Standard functions for data type conversion

9.6.1 Functions for the conversion of numeric data types and bit data types

Explicit conversion is always required if information could be lost, for example, if the value range is decreased or the accuracy is reduced, as is the case for conversion from LREAL to REAL.

The compiler outputs warnings when it detects conversions associated with loss of precision.

Note

The type conversion result may cause errors when the program is running, the error response set in the task configuration will then be triggered, see Execution errors in programs (Page 154).

Special attention is required when converting DWORD to REAL. The bit string from DWORD is taken unchecked as the REAL value. You must make sure that the bit string in DWORD corresponds to the bit pattern of a normalized floating-point number in accordance with IEEE. To do this, you can use the _finite (Page 408) and _isNaN (Page 409) functions.

Otherwise, an error can be triggered (FPU exception (Page 155)) as soon as the REAL value is first used for an arithmetic operation (for example, in the program or when monitoring in the symbol browser).

Explicit data type conversion is performed using standard functions which are listed in the following table.

- **Input parameter**
  Each function for the conversion of a data type has exactly one input parameter named *in*.
- **Return value**
  The result is always the return value of the function. The respective conversion rule is specified in the following table.
- **Names**
  As the data types of the input parameter and the return value come from the respective function name, they are not listed separately in the following table: For example, with the BOOL_TO_BYTE function, the data type of the input parameter is BOOL and the data type of the return value is BYTE.

Table 9-13 Functions for converting numeric data types and bit data types

Function name	Conversion rule	Implicitly possible
BOOL_TO_BYTE	Accept as least significant bit and fill the rest with 0.	Yes
BOOL_TO_DWORD	Accept as least significant bit and fill the rest with 0.	Yes
BOOL_TO_WORD	Accept as least significant bit and fill the rest with 0.	Yes
BOOL_VALUE_TO_DINT	Accept Boolean value as DINT value (0 or 1).	No
BOOL_VALUE_TO_INT	Accept Boolean value as INT value (0 or 1).	No
BOOL_VALUE_TO_LREAL	Accept Boolean value as LREAL value (0.0 or 1.0).	No
BOOL_VALUE_TO_REAL	Accept Boolean value as REAL value (0.0 or 1.0).	No
BOOL_VALUE_TO_SINT	Accept Boolean value as SINT value (0 or 1).	No
BOOL_VALUE_TO_UDINT	Accept Boolean value as UDINT value (0 or 1).	No
BOOL_VALUE_TO_UINT	Accept Boolean value as UINT value (0 or 1).	No
BOOL_VALUE_TO_USINT	Accept Boolean value as USINT value (0 or 1).	No
BYTE_TO_BOOL	Accept the least significant bit.	No
BYTE_TO_DINT	Accept bit string as least significant byte and fill the rest with 0.	No
BYTE_TO_DWORD	Accept bit string as least significant byte and fill the rest with 0.	Yes
BYTE_TO_INT	Accept bit string as least significant byte and fill the rest with 0.	No

Function name	Conversion rule	Implicitly possible
BYTE_TO_SINT	Accept bit string as SINT value.	No
BYTE_TO_UDINT	Accept bit string as least significant byte and fill the rest with 0.	No
BYTE_TO_UINT	Accept bit string as least significant byte and fill the rest with 0.	No
BYTE_TO_USINT	Accept bit string as USINT value.	No
BYTE_TO_WORD	Accept bit string as least significant byte and fill the rest with 0.	Yes
BYTE_VALUE_TO_LREAL	Interpret bit string as USINT value and accept this value.	No
BYTE_VALUE_TO_REAL	Interpret bit string as USINT value and accept this value.	No
DINT_TO_BYTE	Accept the least significant byte as bit string.	No
DINT_TO_DWORD	Accept bit string.	No
DINT_TO_INT	Accept the 2 least significant bytes as INT value.	No
DINT_TO_LREAL	Accept value.	Yes
DINT_TO_REAL	Accept value (accuracy may be lost).	No
DINT_TO_SINT	Accept the least significant byte as SINT value.	No
DINT_TO_UDINT	Accept value as bit string.	No
DINT_TO_UINT	Accept the 2 least significant bytes as UINT value.	No
DINT_TO_USINT	Accept the least significant byte as USINT value.	No
DINT_TO_WORD	Accept the 2 least significant bytes as bit string.	No
DINT_VALUE_TO_BOOL	FALSE (0), if DINT value = 0; else TRUE (1).	No
DWORD_TO_BOOL	Accept the least significant bit.	No
DWORD_TO_BYTE	Accept the least significant byte as bit string.	No
DWORD_TO_DINT	Accept bit string as DINT value.	No
DWORD_TO_INT	Accept the 2 least significant bytes as INT value.	No
DWORD_TO_REAL	Accept bit string as REAL value (validity check of the REAL number is not carried out!). Note the important information at the start of this section!	No
DWORD_TO_SINT	Accept the least significant byte as SINT value.	No
DWORD_TO_UDINT	Accept bit string as UDINT value.	No
DWORD_TO_UINT	Accept the 2 least significant bytes as UINT value.	No
DWORD_TO_USINT	Accept the least significant byte as USINT value.	No
DWORD_TO_WORD	Accept the 2 least significant bytes as bit string.	No
DWORD_VALUE_TO_LREAL	Interpret bit string as UDINT value and accept this value.	No
DWORD_VALUE_TO_REAL	Interpret bit string as UDINT value and accept this value (accuracy can be lost).	No
INT_TO_BYTE	Accept the least significant byte as bit string.	No
INT_TO_DWORD	Accept bit string as least significant word (2 bytes) and fill the rest with 0.	No
INT_TO_DINT	Accept value.	Yes
INT_TO_LREAL	Accept value.	Yes
INT_TO_REAL	Accept value.	Yes
INT_TO_SINT	Accept the least significant byte as SINT value.	No
INT_TO_USINT	Accept the least significant byte as USINT value.	No
INT_TO_UDINT	Accept bit string as least significant word (2 bytes) and fill the rest with 0.	No
INT_TO_UINT	Accept bit string as UINT value.	No

Function name	Conversion rule	Implicitly possible
INT_TO_WORD	Accept bit string.	No
INT_VALUE_TO_BOOL	FALSE (0), if INT value = 0; else TRUE (1).	No
LREAL_TO_DINT	Round off to next integer[1].	No
LREAL_TO_INT	Round off to next integer[1].	No
LREAL_TO_REAL	Accept value (accuracy may be lost).	No
LREAL_TO_SINT	Round off to next integer[1].	No
LREAL_TO_UDINT	Round off value to next integer[1].	No
LREAL_TO_UINT	Round off value to next integer[1].	No
LREAL_TO_USINT	Round off value to next integer[1].	No
LREAL_VALUE_TO_BOOL	FALSE (0), if LREAL value = 0.0; else TRUE (1).	No
LREAL_VALUE_TO_BYTE	Round off value to next integer[1] and accept value as bit string.	No
LREAL_VALUE_TO_DWORD	Round off value to next integer[1] and accept value as bit string (only for positive numbers[2])	No
LREAL_VALUE_TO_WORD	Round off value to next integer[1] and accept value as bit string.	No
REAL_TO_DINT	Round off to next integer[1].	No
REAL_TO_DWORD	Accept bit string.	No
REAL_TO_INT	Round off to next integer[1].	No
REAL_TO_LREAL	Accept value.	Yes
REAL_TO_SINT	Round off to next integer[1].	No
REAL_TO_UDINT	Round off value to next integer[1].	No
REAL_TO_UINT	Round off value to next integer[1].	No
REAL_TO_USINT	Round off value to next integer[1].	No
REAL_VALUE_TO_BOOL	FALSE (0), if REAL value = 0.0; else TRUE (1).	No
REAL_VALUE_TO_BYTE	Round off value to next integer[1] and accept value as bit string.	No
REAL_VALUE_TO_DWORD	Round off value to next integer[1] and accept value as bit string.	No
REAL_VALUE_TO_WORD	Round off value to next integer[1] and accept value as bit string.	No
SINT_TO_BYTE	Accept bit string.	No
SINT_TO_DINT	Accept value.	Yes
SINT_TO_DWORD	Accept bit string as least significant byte and fill the rest with 0.	No
SINT_TO_INT	Accept value.	Yes
SINT_TO_LREAL	Accept value.	No
SINT_TO_REAL	Accept value.	No
SINT_TO_UDINT	Accept bit string as least significant byte and fill the rest with 0.	No
SINT_TO_UINT	Accept bit string as least significant byte and fill the rest with 0.	No
SINT_TO_USINT	Accept bit string.	No
SINT_TO_WORD	Accept bit string as least significant byte and fill the rest with 0.	No
SINT_VALUE_TO_BOOL	FALSE (0), if SINT value = 0; else TRUE (1).	No
StructAlarmId_TO_DINT	Accept bit string.	No
UDINT_TO_BYTE	Accept the least significant byte as bit string.	No
UDINT_TO_DINT	Accept bit string.	No
UDINT_TO_DWORD	Accept bit string.	No
UDINT_TO_INT	Accept the 2 least significant bytes as INT value.	No

Function name	Conversion rule	Implicitly possible
UDINT_TO_LREAL	Accept value.	Yes
UDINT_TO_REAL	Accept value (accuracy may be lost).	No
UDINT_TO_SINT	Accept the least significant byte as SINT value.	No
UDINT_TO_UINT	Accept the 2 least significant bytes as UINT value.	No
UDINT_TO_USINT	Accept the least significant byte as USINT value.	No
UDINT_TO_WORD	Accept the 2 least significant bytes as bit string.	No
UDINT_VALUE_TO_BOOL	FALSE (0), if UDINT value = 0; else TRUE (1).	No
UINT_TO_BYTE	Accept the least significant byte as bit string.	No
UINT_TO_DINT	Accept value.	Yes
UINT_TO_DWORD	Accept bit string as least significant word (2 bytes) and fill the rest with 0.	No
UINT_TO_INT	Accept bit string.	No
UINT_TO_LREAL	Accept value.	No
UINT_TO_REAL	Accept value.	Yes
UINT_TO_SINT	Accept the least significant byte as SINT value.	No
UINT_TO_UDINT	Accept value.	Yes
UINT_TO_USINT	Accept the least significant byte as USINT value.	No
UINT_TO_WORD	Accept bit string.	No
UINT_VALUE_TO_BOOL	FALSE (0), if UINT value = 0; else TRUE (1).	No
USINT_TO_BYTE	Accept bit string.	No
USINT_TO_DINT	Accept value.	No
USINT_TO_DWORD	Accept bit string as least significant byte and fill the rest with 0.	No
USINT_TO_INT	Accept value.	Yes
USINT_TO_LREAL	Accept value.	No
USINT_TO_REAL	Accept value.	No
USINT_TO_SINT	Accept bit string as SINT value.	No
USINT_TO_UDINT	Accept value.	Yes
USINT_TO_UINT	Accept value.	Yes
USINT_TO_WORD	Accept bit string as least significant byte and fill the rest with 0.	No
USINT_VALUE_TO_BOOL	FALSE (0), if USINT value = 0; else TRUE (1).	No
WORD_TO_BOOL	Accept the least significant bit.	No
WORD_TO_BYTE	Accept the least significant byte as bit string.	No
WORD_TO_DINT	Accept bit string as least significant word (2 bytes) and fill the rest with 0.	No
WORD_TO_DWORD	Accept bit string as least significant word (2 bytes) and fill the rest with 0.	Yes
WORD_TO_INT	Accept bit string as INT value.	No
WORD_TO_SINT	Accept the least significant byte as SINT value.	No
WORD_TO_UDINT	Accept bit string as least significant word (2 bytes) and fill the rest with 0.	No
WORD_TO_UINT	Accept bit string.	No
WORD_TO_USINT	Accept the least significant byte as USINT value.	No
WORD_VALUE_TO_LREAL	Interpret bit string as UINT value and accept this value.	No
WORD_VALUE_TO_REAL	Interpret bit string as UINT value and accept this value.	No

[1] If the distance to the two next integers is the same: Round off to next even integer.

[2] The LREAL_VALUE_TO_DWORD function behaves the same as the LREAL_TO_UDINT function. Consequently only positive numbers are converted. If a floating-point number with sign is to be converted to a bit string, the combination of LREAL_TO_DINT and DINT_TO_DWORD must be used.

9.6.2 Functions for converting date and time data types

The following standard functions are available for date and time data types.

For information about arithmetic expressions with date and time data types, see "Arithmetic expressions" in the SIMOTION ST Programming Manual.

Table 9-14 Standard functions for date and time

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
CONCAT_DATE_TOD	in1 in2	DATE TIME_OF_DAY	DATE_AND_TIME	Combine DATE and TIME_OF_DAY to DATE_AND_TIME (DT).
DATE_AND_TIME_TO_TIME_OF_DAY or DT_TO_TOD	in	DATE_AND_TIME	TIME_OF_DAY	Accept time of day.
DATE_AND_TIME_TO_DATE or DT_TO_DATE	in	DATE_AND_TIME	DATE	Accept date.
INT_TO_TIME	in	INT	TIME	Accept value as time (unit is ms) The function does not return any useful results for negative values.
REAL_TO_TIME	in	REAL	TIME	Round off to unsigned integer component and accept the value as indication of time (in ms).
TIME_TO_INT	in	TIME	INT	Accept time (unit is ms) as value.
TIME_TO_REAL	in	TIME	REAL	Accept time (unit is ms) as value.
TIME_TO_UDINT	in	TIME	UDINT	Accept time (unit is ms) as value.
UDINT_TO_TIME	in	UDINT	TIME	Accept value as time (unit is ms). The function does not return any useful results for negative values.

9.6.3 Functions for the conversion of enumeration data types

You can obtain the numeric value of an enumeration data type element with the ENUM_TO_DINT function. When calling the function, you specify the data type of the enumeration element by placing the identifier of the data type and the character # in front of the identifier of the enumeration element (*enum_type#enum_value*).

Table 9-15 Standard functions for date and time

Function name	Input parameter		Return value	Description
	Name	Data type	Data type	
ENUM_TO_DINT	in	any enumeration data type	DINT	Supplies the numeric value of the enumeration element

All user-defined enumeration data types organize the values in ascending order starting with 0.

Enumeration data types that are defined as system data types (e.g. of the SIMOTION devices or technology objects), deviate from this. You will find the values in the relevant List Manuals (reference lists).

9.6.4 Conversions between BYTE and STRING

Description

The implicit conversion from BYTE to STRING and from STRING to BYTE enables:

- A byte to be written to a string or
- A byte to be read from a string (ASCII format, 1 byte per character).

Note

Strings are interpreted as ARRAY[1...stringsize].

Conversion from BYTE to STRING

The conversion is performed through direct assignment of the byte to string element n:

```
mySTring[n] := myByte;
```

Rules:

1. If n > len(myString) and n < maxlen(myString), the length of the string is adjusted. All characters between myString[len(myString)] ... myString[n] are assigned the value "0".
2. If n > maxlen(myString), TSI#ERRNO is set to value 1 (maximum string length exceeded) and myString remains the same.

Conversion from STRING to BYTE

The conversion is performed through direct assignment of string element k to the byte:
`myByte := myString[k];`

Rule

1. If k > len(myString), TSI#ERRNO is set to value 2 (value exceeds the valid range) and myByte is assigned the value 0.

Applications

- Read out of internal variable and, if required, conversion from INT to STRING or DINT to STRING.
- Direct output of STRING on HMI, e.g. WIN CC Op.
- Conversion of STRING to ARRAY OF BYTE and thus output on HMI.

9.6.5 Functions for the conversion of numeric data types to STRING data type

Description

The following functions are used for the conversion of numbers to strings for the purpose of displaying numbers of the DINT/UDINT/REAL/LREAL data types.

Explicit data type conversion is performed using standard functions which are listed in the following table.

- Input parameter
  Each function for the conversion of a data type has exactly one input parameter named *in*.
- Return value
  The result is always the return value of the function.

Rules for the conversion from DINT/UDINT/REAL/LREAL to STRING

- Values are written left-justified in the STRING as decimal number or real number.
- If signs are present, they are written in front of the numerals.
- If the string length is not sufficient, the numeric sequence is truncated on the right (violation of the string length).
- With the conversion to STRING, the number representation is decimal.

Rules for the conversion from STRING to DINT/UDINT/REAL/LREAL

1. Leading white spaces are not taken into account; blanks and tabs are recognized as white spaces.
2. Conversion stops at the end of the string or at the first character that is not a numeral.

3. The function uses the "." as decimal character. If you use a "," as decimal character, the numbers after the comma are not taken into account.
   Example:
   - 1.1 -> STRING_TO_REAL -> 1.1
   - 1.1 -> STRING_TO_REAL -> 1.0
4. If the string does not contain a valid number or the value range is violated, TSI#ERRNO is set to value 3 (invalid number representation) and 0.0 (REAL/LREAL) output.
5. Leading zeros are omitted.

Function	Example	Description
DINT_TO_STRING	myString:=DINT_TO_STRING(myDint) myString := DINT_TO_STRING(mySint)	Observe rules
UDINT_TO_STRING	myString:=UDINT_TO_STRING(myUDint) myString:=UDINT_TO_STRING(myUsint)	
STRING_TO_DINT	myDin t:= STRING_TO_DINT(myString)	A valid number has the form [white space [sign][digits] Decimal numbers are required for conversion from STRING. Octal and hexadecimal notation is not supported.
STRING_TO_UDINT	myUDint:=STRING_TO_UDINT(myString)	
REAL_TO_STRING	myString := REAL_TO_STRING(myReal)	Observe rules
LREAL_TO_STRING	myString := LREAL_TO_STRING(myLReal)	
STRING_TO_REAL	myReal := STRING_TO_REAL(myString)	A valid number has the form [white space [sign][digits][digits][{ e I E } [sign]digits].
STRING_TO_LREAL	myLReal := STRING_TO_LREAL(myString)	

Application

- An HMI fetches texts from the file system (recipe memory) and loads the text via a sequence into the run-time system of SIMOTION (unit variable).
- The data is saved with _saveUnitDataSet or _exportUnitDataSet in the run-time system. Extension of the STRINGs with current SIMOTION data (e.g. actual position).
- The text is output via the serial interface (e.g. ET200).

9.7 Conversion between any data types and defined byte order

9.7.1 General

Variables of any data type (elementary data types, standard data types of technology packages and devices, and user-defined data types) can be converted to the defined big endian or little endian byte order, and vice versa, using the functions specified below:

- To byte arrays of the big endian or little endian type:
  AnyType_to_BigByteArray function, AnyType_to_LittleByteArray function (Page 396)
- From byte arrays of the big endian or little endian type:
  BigByteArray_to_AnyType function, LittleByteArray_to_AnyType function (Page 398)
- To the same data type with big endian or little endian byte order:
  TO_BIG_ENDIAN function, TO_LITTLE_ENDIAN function (Page 401)
- From the same data type with big endian or little endian byte order:
  FROM_BIG_ENDIAN function, FROM_LITTLE_ENDIAN function (Page 402)

For further information (e.g. for the ordering of the byte arrays, application example) see Converting between any data types and byte arrays (marshalling) (Page 501).

These functions are commonly used to create defined transmission formats for data exchange between various devices (see also Communication functions (Page 504)).

9.7.2 AnyType_to_BigByteArray function, AnyType_to_LittleByteArray function

Description

The functions convert a variable of any data type (elementary data types, user-defined data types, system data types of technology packages, and devices) to a byte array.

- For AnyType_to_BigByteArray:
  Big endian-type byte array (most significant byte at low memory address)
- For AnyType_to_LittleByteArray:
  Little endian-type byte array (least significant byte at low memory address)

The array index of the first element to be occupied in the array is an optional constant offset (default = 0). It must fall within the array limits.

When an ST source file is compiled, a check is made to determine whether the offset falls within the array limits and whether the variable can be displayed completely in the byte array (between the offset and the upper array limit).

Only those byte array elements that are covered by variables to be converted are occupied with values. Other elements of the byte array remain unchanged.

Note

Where possible, the function call and the processing of the result should be made in the same task.

If the function call and the processing of the result are made in different tasks, they must be synchronized using appropriate means (e.g. _testAndSetSemaphore (Page 418), _releaseSemaphore (Page 419)). Otherwise data consistency is not guaranteed and undefined values can result.

If the return value of the function should be assigned to a global variable that is evaluated in a higher priority task, the following procedure is recommended (also for simple data types):

1. Conversion to a temporary variable.
2. Assign the temporary variable to the global target variable.

Note

Each variable of data type BOOL (including variables that are components within a structured data type) occupies one byte in the byte array.

For the big endian and little endian byte order as well as the use of the functions, see Converting between any data types and byte arrays (marshalling) (Page 501).

Declaration

```
FUNCTION AnyType_to_BigByteArray (
        anydata : ANY,         // Any data type
        { offset  : DINT         // Only constants permitted
        }
        ) : ARRAY [..] OF BYTE       // Big endian
FUNCTION AnyType_to_LittleByteArray (
        anydata : ANY,         // Any data type
        { offset  : DINT         // Only constants permitted
        }
        ) : ARRAY [..] OF BYTE  // Little endian
```

Input parameter

anydata

Data type: ANY

Variable of any data type.

The following data types are permitted:

- Elementary data types (ANY_ELEMENTARY)
- User-defined data types (UDT)
- System data types of technology packages and devices

The following data types are **not** permitted:

- Technology object data types
- Object-oriented interfaces
- Structures with overlapping memory areas (STRUCT OVERLAP)
  Components within these structures are, however, permitted.

offset (optional)

Data type: DINT

Default 0

Constant, specifies the first element to be assigned in the array.

Return value

Data type: ARRAY [..] OF BYTE

- For AnyType_to_BigByteArray:
  In big endian order (most significant byte at low memory address)
- For AnyType_to_LittleByteArray:
  In little endian order (least significant byte at low memory address)

9.7.3 BigByteArray_to_AnyType function, LittleByteArray_to_AnyType function

Description

The functions convert a byte array to a variable of any data type (elementary data types, user-defined data types, system data types of technology packages and devices, data types of technology objects).

- For BigByteArray_to_AnyType
  Big endian-type byte array (most significant byte at low memory address)
- For LittleByteArray_to_AnyType
  Little endian-type byte array (least significant byte at low memory address)

The array index of the first element to be evaluated in the array is an optional constant offset (default = 0). It must fall within the array limits.

When an ST source file is compiled, a check is made to determine whether the offset falls within the array limits and whether the byte array (between the offset and the upper array limit) completely covers the variable.

Note

Where possible, the function call and the processing of the result should be made in the same task.

If the function call and the processing of the result are made in different tasks, they must be synchronized using appropriate means (e.g. _testAndSetSemaphore (Page 418), _releaseSemaphore (Page 419)). Otherwise data consistency is not guaranteed and undefined values can result.

If the return value of the function should be assigned to a global variable that is evaluated in a higher priority task, the following procedure is recommended (also for simple data types):

1. Conversion to a temporary variable.
2. Assign the temporary variable to the global target variable.

Note

One byte from the byte array is assigned to each variable of data type BOOL (including variables that are components within a structured data type).

For the big endian and little endian byte order as well as the use of the functions, see Converting between any data types and byte arrays (marshalling) (Page 501).

Declaration

```
FUNCTION BigByteArray_to_AnyType (
    byteArray : ARRAY [..] OF BYTE, // Big endian
    { offset  : DINT        // Only constants permitted
    }
    ) : ANY
FUNCTION LittleByteArray_to_AnyType (
    byteArray : ARRAY [..] OF BYTE, // Little endian
    { offset  : DINT        // Only constants permitted
    }
    ) : ANY
```

Input parameter

byteArray

Data type: ARRAY [..] OF BYTE

- For BigByteArray_to_AnyType
  In big endian order (most significant byte at low memory address)
- For LittleByteArray_to_AnyType
  In little endian order (least significant byte at low memory address)

offset (optional)

Data type:	DINT
Default	0

Constant, specifies the first element to be assigned in the array.

Return value

Data type: ANY

Any data type.

The following data types are permitted:

- Elementary data types (ANY_ELEMENTARY)
- User-defined data types (UDT)
- System data types of technology packages and devices

The following data types are **not** permitted:

- Technology object data types
- Object-oriented interfaces
- Structures with overlapping memory areas (STRUCT OVERLAP)
  Components within these structures are, however, permitted.

Note

The marshalling function result may cause errors when the program is running, the error response set in the task configuration will then be triggered, see Execution errors in programs (Page 154).

Proceed with caution when converting byte arrays to the general ANY_REAL data type or to structures that contain this data type. The bit string from the byte array is taken unchecked as the ANY_REAL value. You must make sure that the bit string of the byte array corresponds to the bit pattern of a normalized floating-point number in accordance with IEEE 754. To do this, you can use the _finite (Page 408) and _isNaN (Page 409) functions or the IS_VALID (Page 410) function.

Otherwise, an error can be triggered (FPU exception (Page 155)) as soon as the ANY_REAL value is first used for an arithmetic operation (for example, in the program or when monitoring in the symbol browser).

9.7.4 TO_BIG_ENDIAN function, TO_LITTLE_ENDIAN function

Description

The functions store a variable of any data type (elementary data types, user-defined data types, system data types of technology packages, and devices) in a variable of the same byte array data type. Whereby, elementary data types that comprise several bytes are stored as follows:

- For TO_BIG_ENDIAN:
  In big endian byte order (most significant byte at low memory address)
- With TO_LITTLE_ENDIAN:
  In little endian byte order (least significant byte at low memory address)

To use this function, you must activate the "Permit language extensions IEC61131-3rd Edition" compiler option, see SIMOTION ST Programming and Operating Manual.

Note

Where possible, the function call and the processing of the result should be made in the same task.

If the function call and the processing of the result are made in different tasks, they must be synchronized using appropriate means (e.g. _testAndSetSemaphore (Page 418), _releaseSemaphore (Page 419)). Otherwise data consistency is not guaranteed and undefined values can result.

If the return value of the function should be assigned to a global variable that is evaluated in a higher priority task, the following procedure is recommended (also for simple data types):

1. Conversion to a temporary variable.
2. Assign the temporary variable to the global target variable.

Note

Each variable of data type BOOL (including variables that are components within a structured data type) occupies one byte in the byte array.

For information about big endian and little endian byte ordering, see Converting between any data types and byte arrays (marshalling) (Page 501).

Declaration

```
FUNCTION TO_BIG_ENDIAN (
        in  : ANY,        // Any data type
        )   : ANY       // Big endian
FUNCTION TO_LITTLE_ENDIAN (
        in  : ANY,        // Any data type
        )   : ANY       // Little endian
```

Input parameter

in

Data type: ANY

Variable of any data type.

The following data types are permitted:

- Elementary data types (ANY_ELEMENTARY)
- User-defined data types (UDT)
- System data types of technology packages and devices

The following data types are **not** permitted:

- Technology object data types
- Object-oriented interfaces
- Structures with overlapping memory areas (STRUCT OVERLAP)
  Components within these structures are, however, permitted.

Return value

Data type: ANY

The same data type as the *in* input parameter.

- For TO_BIG_ENDIAN:
  Elementary data types that comprise several bytes to the big endian byte order (most significant byte at low memory address)
- For TO_LITTLE_ENDIAN:
  Elementary data types that comprise several bytes to the little endian byte order (least significant byte at low memory address)

9.7.5 FROM_BIG_ENDIAN function, FROM_LITTLE_ENDIAN function

Description

The functions store a variable of any data type (elementary data types, user-defined data types, system data types of technology packages and devices, data types of technology objects) in a variable of the same data type. Whereby, elementary data types in the input parameter that comprise several bytes have the following byte order:

- For FROM_BIG_ENDIAN:
  The big endian byte order (most significant byte at low memory address)
- For FROM_LITTLE_ENDIAN:
  The little endian byte order (least significant byte at low memory address)

To use this function, you must activate the "Permit language extensions IEC61131-3rd Edition" compiler option, see SIMOTION ST Programming and Operating Manual.

Note

Where possible, the function call and the processing of the result should be made in the same task.

If the function call and the processing of the result are made in different tasks, they must be synchronized using appropriate means (e.g. _testAndSetSemaphore (Page 418), _releaseSemaphore (Page 419)). Otherwise data consistency is not guaranteed and undefined values can result.

If the return value of the function should be assigned to a global variable that is evaluated in a higher priority task, the following procedure is recommended (also for simple data types):

1. Conversion to a temporary variable.
2. Assign the temporary variable to the global target variable.

Note

One byte from the byte array is assigned to each variable of data type BOOL (including variables that are components within a structured data type).

For information about big endian and little endian byte ordering, see Converting between any data types and byte arrays (marshalling) (Page 501).

Declaration

```
FUNCTION FROM_BIG_ENDIAN (
    in : ANY // Any data type, big endian
    )  : ANY
FUNCTION FROM_LITTLE_ENDIAN (
    in : ANY // Any data type, little endian
    )  : ANY
```

Input parameter

in

Data type: ANY

Any data type.

- For FROM_BIG_ENDIAN
  Elementary data types that comprise several bytes to the big endian byte order (most significant byte at low memory address)
- For FROM_LITTLE_ENDIAN
  Elementary data types that comprise several bytes to the little endian byte order (least significant byte at low memory address)

The following data types are permitted:

- Elementary data types (ANY_ELEMENTARY)
- User-defined data types (UDT)
- System data types of technology packages and devices

The following data types are **not** permitted:

- Technology object data types
- Object-oriented interfaces
- Structures with overlapping memory areas (STRUCT OVERLAP)
  Components within these structures are, however, permitted.

Return value

Data type: ANY

The same data type as the *in* input parameter.

9.8 Combining bit-string data types

9.8.1 General information for combining bit-string data types

The following functions enable several bit-string data-type variables to be combined into a higher-level data-type variable.

The inverse functions are implemented as function blocks, see Separating bit-string data types (Page 539).

9.8.2 _BYTE_FROM_8BOOL function

Description

This function combines eight variables of data type BOOL into one variable of data type BYTE.

Declaration

```
FUNCTION _BYTE_FROM_8BOOL (
       { bit0,         // Least significant bit
         bit1, bit2, bit3, bit4, bit5, bit6,
         bit7 : BOOL   // Most significant bit
         }
    ) : BYTE
```

Input parameter

bit0 (optional)
...
bit7 (optional)

Data type: BOOL
Default FALSE

Up to eight variables of data type BOOL which are to be combined into a variable of data type BYTE

bit0: Least significant bit
...
bit7: Most significant bit

Return value

Data type: BYTE
Byte formed by combining input parameters.

9.8.3 _WORD_FROM_2BYTE function

Description

This function combines two variables of data type BYTE into one variable of data type WORD.

Declaration

```
FUNCTION _WORD_FROM_2BYTE (
       { byte0,          // Less significant byte
         byte1   : BYTE   // More significant byte
         }
    ) : WORD
```

Input parameter

byte0 (optional)
byte1 (optional)

Data type: BYTE
Default BYTE#0

Up to two variables of data type BYTE, which are to be combined into a variable of data type WORD

byte0: Less significant byte
byte1: More significant byte

Return value

Data type: WORD
Word formed by combining input parameters.

9.8.4 _DWORD_FROM_2WORD function

Description

This function combines two variables of data type WORD into one variable of data type DWORD.

Declaration

```
FUNCTION _DWORD_FROM_2WORD (
       { word0,          // Less significant word
         word1 : WORD    // More significant word
         }
    ) : DWORD
```

Input parameter

word0 (optional)
word1 (optional)

Data type: WORD
Default WORD#0

Up to two variables of data type WORD, which are to be combined into a variable of data type DWORD

word0: Less significant word
word1: More significant word

Return value

Data type: DWORD
Double word formed by combining input parameters.

9.8.5 _DWORD_FROM_4BYTE function

Description

This function combines four variables of data type BYTE to one variable of data type DWORD.

Declaration

```
FUNCTION _DWORD_FROM_4BYTE (
       { byte0          // Least significant byte
         byte1, byte2,
         byte3 : BYTE   // Most significant byte
         }
    ) : DWORD
```

Input parameter

byte0 (optional)
byte1 (optional)
byte2 (optional)
byte3 (optional)

Data type: BYTE
Default BYTE#0

Up to four variables of data type BYTE, which are to be combined into a variable of data type DWORD

byte0: Least significant byte
...
byte3: Most significant byte

Return value

Data type: DWORD
Double word formed by combining input parameters.

9.9 Conversion of technology object data types

9.9.1 AnyObject_to_Object function

Description

The function converts variables of a hierarchical TO data type (driveAxis, posAxis, followingAxis) or of the general ANYOBJECT type to a compatible TO data type.

You will find examples and additional information in Conversion of TO data types (Page 141).

Declaration

```
FUNCTION AnyObject_to_Object (
        in    : ANYOBJECT
    ) : ANYOBJECT
```

Input parameter

in

Data type: ANYOBJECT

Variable of a TO data type (or ANYOBJECT) or a TO instance

Return value

Data type: ANYOBJECT

Value is TO#NIL, if type conversion is not possible

9.10 Functions for verification of floating-point numbers

9.10.1 _finite function

Description

The function checks whether the input parameter complies with the bit pattern for infinite in accordance with IEEE 754.

In combination with the _isNaN (Page 409) function, it is used in particular to check whether bit strings converted to floating-point numbers correspond to the bit pattern of a normalized floating-point number in accordance with IEEE 754.

This prevents the error response specified during task configuration from being triggered (see "Execution errors in programs" (Page 154)) as soon as an invalid floating-point number is used for an arithmetic operation for the first time (for example in the program or when monitoring in the symbol browser).

Declaration

```
_finite (
        in : ANY_REAL
     ) : BOOL
```

Input parameter

in

Data type: ANY_REAL
Variable to be checked

Return value

Data type:	BOOL
FALSE	Bit pattern for infinite in accordance with IEEE 754
TRUE	No bit pattern for infinite in accordance with IEEE 754, i.e. valid floating-point number within the value range or invalid bit pattern (NaN Not a Number)

Example

```
var_real := DWORD_TO_REAL (var_dword);
IF NOT _finite (var_real) OR _isNaN (var_real) THEN
    ; // Error handling
ELSE
    var_real := SQRT (var_real);
END_IF;
```

9.10.2 _isNaN function

Description

The function verifies whether the input parameter corresponds to an invalid bit pattern of a floating-point number in accordance with IEEE 754 (is Not a Number NaN).

In combination with the _finite (Page 408) function, it is used in particular to check if bit strings converted to floating-point numbers correspond to the bit pattern of a normalized floating-point number in accordance with IEEE 754.

This prevents the error response specified during task configuration from being triggered (see "Execution errors in programs" (Page 154)) as soon as an invalid floating-point number is used for an arithmetic operation for the first time (for example in the program or when monitoring in the symbol browser).

Declaration

```
_isNaN (
        in : ANY_REAL
     ) : BOOL
```

Input parameter

in

Data type: ANY_REAL
Variable to be checked

Return value

Data type:	BOOL
FALSE	Valid bit pattern or bit pattern for infinite in accordance with IEEE 754
TRUE	Invalid bit pattern in accordance with IEEE 754 (NaN Not a Number)

Example

```
var_real := DWORD_TO_REAL (var_dword);
IF NOT _finite (var_real) OR _isNaN (var_real) THEN
    ; // Error handling
ELSE
    var_real := SQRT (var_real);
END_IF;
```

9.10.3 IS_VALID function

Description

The function checks whether the input parameter complies with the bit pattern of a normalized floating-point number in accordance with IEEE 754.

This prevents the error response specified during task configuration from being triggered (see "Execution errors in programs" (Page 154)) as soon as an invalid floating-point number is used for an arithmetic operation for the first time (for example in the program or when monitoring in the symbol browser).

To use this function, you must activate the "Permit language extensions IEC61131-3rd Edition" compiler option, see SIMOTION ST Programming and Operating Manual.

Declaration

```
IS_VALID (
        in : ANY_REAL
     ) : BOOL
```

Input parameter

in

Data type: ANY_REAL

Variable to be checked

Return value

Data type:	BOOL
FALSE	Bit pattern for infinite in accordance with IEEE 754 or invalid bit pattern (NaN Not a Number)
TRUE	Valid floating-point number within the value range

Example

```
var_real := DWORD_TO_REAL (var_dword);
IF IS_VALID (var_real) THEN
    var_real := SQRT (var_real);
ELSE
    ; // Error handling
END_IF;
```

9.11 Functions for selection

9.11.1 SEL function

Description

Binary selection

The return value is one of the input parameters in0 or in1, depending on the value of input parameter g.

Declaration

```
FUNCTION SEL (
        g    : BOOL,
        in0  : ANY,
        in1  : ANY
        ) : ANY
```

Input parameter

g

Data type: BOOL

Selection of the input parameter in0 or in1

in0, in1

Data type: ANY

The input parameters in0 and in1 must either be of the same data type or must be convertible to a common data type by implicit conversion (see "Elementary data type conversion" in the SIMOTION ST Programming Manual).

Return value

Data type: ANY

Selected input parameter

in0 If g = 0 (FALSE)

in1 If g = 1 (TRUE)

The data type corresponds to the common data type of the input parameters in0 and in1.

9.11.2 MUX function

Description

Expandable multiplex function

The return value is one of the *n* input parameters in0 to in*n*-1, depending on the value of input parameter k.

Note

This description is not relevant to LAD/FDB. The corresponding function is described in the SIMOTION LAD/FDB Programming Manual.

Declaration

```
FUNCTION MUX (
        k       : ANY_INT,
        in0     : ANY,
        ...
        inn-1   : ANY
        ) : ANY
```

Input parameter

k

Data type: ANY_INT

Selection of the input parameter in0 to in*n*-1.

The value range depends on the number n of the input parameters in0...in*n*-1: 0 ≤ k ≤ *n*-1.

If illegal values are specified, input parameter in0 will be selected.

in0

...

in*n*–1

Data type: ANY

The number n of the input parameters in0 and in*n*-1 may vary.

All input parameters in0 and in*n*-1 must either be of the same data type or must be convertible to a common data type by implicit conversion (see "Elementary data type conversion" in the SIMOTION ST Programming Manual).

If *n* formal parameters in0 to in*n*-1 are specified when calling the function, this must be done in ascending, uninterrupted order; for four input parameters, for example, the identifiers of the formal parameters are: in0, in1, in2, in3.

Return value

Data type: ANY

Input parameter in*m* if input parameter k has the value *m*.

The data type corresponds to the common data type of the input parameters in0 to in*n*-1.

9.11.3 MAX function

Description

Expandable maximum function

The return value is the maximum value of *n* input parameters in0 to in*n*-1.

Note

This description is not relevant to LAD/FDB. The corresponding function is described in the SIMOTION LAD/FDB Programming Manual.

Declaration

```
FUNCTION MAX (
        in0    : ANY_ELEMENTARY,
        ...
        inn-1  : ANY_ELEMENTARY
        ) : ANY_ELEMENTARY
```

Input parameter

in0
...
in*n*–1

Data type: ANY_ELEMENTARY

The number n of the input parameters in0 and in*n*-1 may vary.

All input parameters in0 and in*n*-1 must either be of the same data type or must be convertible to the most powerful data type by implicit conversion (see "Elementary data type conversion" in the SIMOTION ST Programming Manual).

If *n* formal parameters in0 to in*n*-1 are specified when calling the function, this must be done in ascending, uninterrupted order; for four input parameters, for example, the identifiers of the formal parameters are: in0, in1, in2, in3.

Return value

Data type: ANY_ELEMENTARY

Maximum of the input parameters

The data type corresponds to the most powerful data type of the input parameters in0 to in*n*-1.

9.11.4 MIN function

Description

Expandable minimum function

The return value is the minimum value of *n* input parameters in0 to in*n*-1.

Note

This description is not relevant to LAD/FDB. The corresponding function is described in the SIMOTION LAD/FDB Programming Manual.

Declaration

```
FUNCTION MIN (
        in0    : ANY_ELEMENTARY,
        ...
        inn-1  : ANY_ELEMENTARY
        ) : ANY_ELEMENTARY
```

Input parameter

in0

...

in*n*–1

Data type: ANY_ELEMENTARY

The number n of the input parameters in0 and in*n*-1 may vary.

All input parameters in0 and in*n*-1 must either be of the same data type or must be convertible to the most powerful data type by implicit conversion (see "Elementary data type conversion" in the SIMOTION ST Programming Manual).

If *n* formal parameters in0 to in*n*-1 are specified when calling the function, this must be done in ascending, uninterrupted order; for four input parameters, for example, the identifiers of the formal parameters are: in0, in1, in2, in3.

Return value

Data type: ANY_ELEMENTARY

Minimum of the input parameters

The data type corresponds to the most powerful data type of the input parameters in0 to in*n*-1.

9.11.5 LIMIT function

Description

Limiting function

The input parameter *in* is limited to values between the lower limiting value *mn* and the upper limiting value *mx*.

The function value is calculated as follows with the MIN (Page 415) and MAX (Page 414) selection functions:

```
LIMIT = MIN (MAX (in, mn), mx)
```

Declaration

```
FUNCTION LIMIT (
        mn  : ANY_ELEMENTARY,
        in  : ANY_ELEMENTARY,
        mx  : ANY_ELEMENTARY
        ) : ANY_ELEMENTARY
```

Input parameter

mn

Data type: ANY_ELEMENTARY

Lower limiting value

All input parameters must either be of the same data type or must be convertible to the most powerful data type by implicit conversion (see "Elementary data type conversion" in the SIMOTION ST Programming Manual).

in

Data type: ANY_ELEMENTARY

Value to be limited

All input parameters must either be of the same data type or must be convertible to the most powerful data type by implicit conversion (see "Elementary data type conversion" in the SIMOTION ST Programming Manual).

mx

Data type: ANY_ELEMENTARY

Upper limiting value

All input parameters must either be of the same data type or must be convertible to the most powerful data type by implicit conversion (see "Elementary data type conversion" in the SIMOTION ST Programming Manual).

Return value

Data type: ANY_ELEMENTARY

MIN (MAX (in, mn), mx)

The data type corresponds to the most powerful data type of the input parameters.

9.12 Consistent access to global variables of derived data types (UDT)

9.12.1 General

When accessing **global** variables of derived data types (see **User-defined data types (UDT)** in the ST programming manual), the user is responsible for ensuring data consistency when multiple tasks access the same user variables (I/O variables, system variables, system variables of technology objects, global device variables, and global unit variables, see **Variable model** in the programming manuals).

Note

Consistent data access is always ensured within a task.

You can work with semaphores to ensure that global variables of derived data types (UDT) are written and read consistently.

A global variable of data type DINT serves as a semaphore. Use the following functions to change and test the status of the semaphore:

- _testAndSetSemaphore (Page 418)
- _releaseSemaphore (Page 419)

Consistent data access to global variables is ensured under the following conditions:

1. All tasks signal access to global variables by setting a semaphore.
2. All tasks access global variables only when the semaphore is enabled.

For more information on consistent data access and semaphores, refer to “Consistent reading and writing of variables (semaphores)” (Page 478).

9.12.2 _testAndSetSemaphore function

Description

This function checks whether the semaphore is set.

The semaphore is always set when the function ends. Additional calls of this function (even from other programs) return a value of FALSE until the _releaseSemaphore (Page 419) function is called.

Declaration

```
_testAndSetSemaphore     (
                          sema    : DINT
                         ) : BOOL
```

Input parameter

sema

Data type DINT

Sema is a **global** variable of the DINT data type; it serves as semaphore. It must not be indexed. If it is an element of an array, the index must be specified when compiling (e.g. a[2]).

Return value

Data type: BOOL

This return value determines whether the semaphore is set:

TRUE	The semaphore was free and was successfully set/assigned by this call. If TRUE, the semaphore must be released again after completion of the critical code section by calling the **_releaseSemaphore** function so it can be assigned at another location.
FALSE	The semaphore was already set/assigned by another program.

Example

See Example: Consistent data access with semaphores (Page 480).

9.12.3 _releaseSemaphore function

Description

This function releases the semaphore. The next call of the _testAndSetSemaphore (Page 418) function (even from different programs) results in a return value of TRUE.

Declaration

```
_releaseSemaphore     (
                       sema    : DINT
                       ) : VOID
```

Input parameter

sema

Data type: DINT

Sema is a **global** variable of the DINT data type; it serves as semaphore. It must not be indexed. If it is an element of an array, the index must be specified when compiling (e.g. a[2]).

Return value

none

Example

See Example: Consistent data access with semaphores (Page 480).

9.13 Access to system variables and inputs/outputs

9.13.1 General information on accessing system variables and inputs/outputs

The _getSafeValue and _setSafeValue functions allow a special error response for access to system variables, configuration data, or I/O variables. It is possible to specify a different response from what has been configured in the event of an error (see Access errors to system variables and configuration data, as well as I/O variables for direct access (Page 156)).

The system functions _getSafeValue and _setSafeValue require a lot of execution time. Therefore, only use them when necessary, e.g. in an IF statement, to wait for the restart of a TO.

With V4.1 and higher, you can also specify a "replacement value" or "last value" for accesses to system variables and config data (V4.1.3 and higher) in the event of an error (e.g. TO is in restart). The configuration data restartInfo.behaviorInvalidSysvarAccess is relevant for this. See System variables (Page 143) or Configuration data (Page 147).

The _getInOutByte function enables direct read access to individual I/O bytes by specifying the address.

See also

Access errors to system variables and configuration data, as well as I/O variables for direct access (Page 156)

System variables (Page 143)

Configuration data (Page 147)

9.13.2 _getSafeValue function

This function reads the specified system variable (or configuration data element) or I/O variable and returns the value in another variable.

When this function is used (instead of a variable assignment), the transition to STOP mode can be prevented if an error occurs when accessing system variables, configuration data or I/O variables (e.g. while restarting a technology object, or in the event of an I/O failure).

Note

Runtime of the function

- Up to SIMOTION Kernel V4.3
  The runtime can be very long. Therefore, this function is not suitable for use in fast cyclic tasks.
- As of SIMOTION Kernel V4.4
  The runtime has been optimized. Access from synchronous user tasks is possible.

Specifying the error response

You can control the error response using the *accessmode* parameter:

- CONFIGURED (default): The error response defined by *restart.behaviorInvalidSysvarAccess* is used, see Access errors to system variables and configuration data, as well as I/O variables for direct access (Page 156).
- NO_CHANGE:
  - System variables and configuration data:
    Variable is not read and the value remains undefined, see System variables (Page 143).
  - I/O variables:
    With read access (to inputs or outputs): The last valid value is applied.
- DEFAULT_VALUE: Substitute value or limit value is read.
- STOP_DEVICE: SIMOTION device switches to STOP mode. When a transition to STOP mode occurs, the ExecutionFaultTask is not started.

You can use the return value to determine whether the access was successful.

Declaration

```
_getSafeValue (
         variable      : ANY,        //System variable,
                                     // Configuration data or
                                     // I/O variable
         accessMode    : EnumAccessMode,
         getValue      : ANY
         )             : EnumSetAndGetSafeValue
```

Input parameter

variable

Data type: ANY

System variable, configuration data item or I/O variable to be read.

accesssMode

Data type: EnumAccessMode

Response when an error occurs during read access.

```
TYPE EnumAccessMode : (
// Response as configured:
    CONFIGURED  // System variables and config data:
                // Behavior depends on config data item
                // restartInfo.behaviorInvalidSysvarAccess
                // For I/O variables:
                // Strategy specified during creation
                // of the I/O variables
// Different response than configured:
    NO_CHANGE      // For system variables and
                   // configuration data: Variable
                   // do not read.
                   // For I/O variables: Accept
                   // last available (valid)
                   // value.
    DEFAULT_VALUE  // Configured value or
                   // Substitute value for I/O variables
    STOP_DEVICE    // Device switches to STOP mode.
END_TYPE
```

getvalue

Data type: ANY

Name of the variable to which the current value of the system variable (or configuration data item) or I/O variable is written.

The following applies for the data type:

- It must be the same as the data type of the variable to be read, or
- The data type of the variable to be read must be able to be converted to this data type by means of implicit type conversion.

Return value

Data type: EnumSetAndGetSafeValue

The return value indicates whether the access was successful.

```
TYPE EnumSetAndGetSafeValue : (
 // Access successful:
 OK            // Access was successful.
 // Access error:
, NO_CHANGE  // For system variables and configuration data: Variable was
             // not read, last value
             // For I/O variables: Last
             // available (valid) value was
             // accepted.
, DEFAULT_VALUE    // Only for I/O variables:
                   // Substitute value was accepted.
, INVALID_VALUE    // Only for configuration data:
                   // Configuration data item was
                   // not read, default value is returned,
                   // impermissible parameter
                   //(accessMode = DEFAULT_VALUE).
, NO_ACCESS )      // Only for configuration data:
                   // Configuration data item was
                   // not read, default value is returned
END_TYPE
```

See also

Errors when accessing system data with _get/_setSafeValue (Page 193)

9.13.3 _setSafeValue function

The function behaves differently as of V4.1.3. It writes the specified value to the system variable, configuration data item or I/O variable and there is an option for it to return the currently written value in another variable. The response in the event of a fault can be specified differently from the configured response.

When this function is used (instead of a variable assignment), the transition to STOP mode can be prevented if an error occurs when accessing system variables, configuration data or I/O variables (e.g. while restarting a technology object, or in the event of an I/O failure).

Note

Runtime of the function

- Up to SIMOTION Kernel V4.3
  The runtime can be very long. Therefore, this function is not suitable for use in fast cyclic tasks.
- As of SIMOTION Kernel V4.4
  The runtime has been optimized. Access from synchronous user tasks is possible.

Specifying the error response

You can control the error response using the *accessmode* parameter:

- CONFIGURED (default): The error response specified in *restart.behaviorInvalidSysvarAccess* is used, see Errors when accessing system variables and configuration data as well as I/O variables for direct access (Page 156).
- NO_CHANGE:
  - System variables and configuration data:
    Value is not written and remains unchanged or retains the last available value.
  - I/O variables:
    With write access (to outputs): The value is written to the variable. However, it will not be active at the output until the output becomes available again.
- DEFAULT_VALUE: Substitute value or limit value is written.
- STOP_DEVICE: SIMOTION device switches to STOP mode. When a transition to STOP mode occurs, the ExecutionFaultTask is not started.
- Value range for variable

The value is restricted to the value range and is not affected by the accessMode parameter.

You can use the return value to determine whether the access was successful.

Declaration

```
_setSafeValue (
        variable     : ANY,         //System variable,
                                   // Configuration data or
                                   // I/O variable
        value        : ANY,
        accessMode   : EnumAccessMode,
        setValue     : ANY
         ) : EnumSetAndGetSafeValue
```

Input parameter

variable

Data type: ANY

System variable, configuration data item or I/O variable to be written.

value

Data type**:** ANY

Value to be written to the system variable (or configuration data item) or I/O variable. The value data type must be the same as the data type of the variable to be written to, or it must be able to be converted to this data type by means of implicit type conversion.

accessMode

Data type: EnumAccessMode

Response when an error occurs during write access:

```
TYPE EnumAccessMode : (
 // Response as configured:
     CONFIGURED       // System variables and config data:
                       // Behavior depends on config data item
                       // restartInfo.behaviorInvalidSysvarAccess
                       // For I/O variables:
                       // Strategy specified during creation
                       // of the I/O variables
 // Different response than configured:
    , NO_CHANGE        // For system variables and
                       // configuration data: Current value
                       // Is not written.
                       //For I/O variables
                       // The value transferred in the
                       // parameter is written to the
                       // written. It is not active at the output
                       // until the output is
                       // again available.
, DEFAULT_VALUE
                       // For system variables and config data:
                       // Variable is not described.
                       // For I/O variables:
                       // Variable is described with the
                       // substitute value specified
                       // when creating the variable.
, STOP_DEVICE )        // Device goes into STOP
                       // mode
END_TYPE
```

setvalue

Data type: ANY

Name of a variable to which the current value of the system variable, configuration data item, or I/O variable is written.

The following applies for the data type:

- It must be of the same data type as the variable to be written, or
- The data type of the variable to be written must be able to be converted to this data type by means of implicit type conversion.

If, for example, 100 is to be written, but only 90 was entered, *setValue* is given a value of 90.

Return value

Data type: EnumSetAndGetSafeValue

The return value indicates whether the access was successful.

```
TYPE EnumSetAndGetSafeValue : (
 // Access successful:
 OK                // Access was successful.
 // Access error:
, NO_CHANGE        // For system variables and config data:
                   // Value is not written.
                   // For I/O variables: The value transferred
                   // in the parameter was
                   // written. It is not active at the output
                   // until the output is
                   // again available.
, DEFAULT_VALUE    // For system variables: Limitation
                   // active, limit value was
                   // written.
                   // For I/O variables: Substitute value was
                   // written. It is not active at the output
                   // until the output is
                   // again available.
, NO_ACCESS )     // Only for configuration data and system variables:
                   // Value of the configuration data item was
                   // not changed,
                   // Configuration data item
                   // not available).
END_TYPE
```

See also

Errors when accessing system data with _get/_setSafeValue (Page 193)

9.13.4 _getInOutByte function

This function enables direct read access to individual I/O bytes through specification of the input/output address.

In the event of an access error, the PeripheralFaultTask is not called; rather a corresponding value is returned in the functionResult component of the return value.

If an I/O variable (for direct access or the process image of the cyclic task) (see **Direct access and process image of the cyclical tasks** in the ST Programming Manual) is defined for the specified I/O address and a replacement value is specified, this replacement value is returned in the event of an access error.

Evaluation of the return value can determine, for example, whether an input or an output is assigned to the address in the hardware.

Note

The runtime of this function can be very long. Therefore, this function is not suitable for use in fast cyclic tasks.

Declaration

```
_getInOutByte (
            mode          : EnumInOutDirection,
            logAddress    : DINT
         )                 : StructRetGetInOutByte
```

Input parameters

mode

Data type: EnumInOutDirection

Specifies whether logAddress is used to access the input or output.

```
TYPE EnumInOutDirection : (
      IN            // Access to input
 ,    OUT )        // Access to output
END_TYPE
```

logAddress

Data type: DINT

Logic address of input or output

Return value

Data type: StructRetGetInOutByte
Function call result and read byte value

```
TYPE StructRetGetInOutByte : STRUCT
    functionResult    : DINT;    // Result of the
                                 // function call
    value             : BYTE;    // Value of byte read
 END_STRUCT;
END_TYPE
```

Possible values of functionResult (result of function call):

0	No error, access okay.
16#FFFF_FFFA	Not enough memory available
16#FFFF_FFFE	Access error
16#FFFF_FFFF	Input/output not available

9.14 Backing up data from the user program

9.14.1 General information on saving data sets from the user program

The functions below are for:

- Saving, loading, or initializing the following values:
  - Non-retentive or retentive global unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit)
  - Non-retentive or retentive global device variables (declared via project navigator)

  Data backup is performed by selecting the relevant function
  - Binary: The backed-up data set can no longer be read after the version ID of the data segment has been changed.
  - In XML format: The backed-up data set can be read after the version ID of the data segment has been changed.
- Managing the data set in which the values are backed up

The values are backed up in a data set that is identified uniquely by specifying the storage location, the name of the unit, and a data set number.

The application of these functions - in particular, the step enabling condition - is described in detail in Data backup and initialization from user program (Page 481).

As well as saving retentive variables as individual data sets, it is also possible to back up all the retentive data to the memory card using the _savePersistentMemoryData function. For more information, see the *System Functions/Variable Devices* Parameter Manual.

9.14.2 _saveUnitDataSet function

Description

The values of the following variables are saved as a binary data set:

- Non-retentive or retentive global unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit, LAD/FBD unit)
- Non-retentive or retentive global device variables (declared via project navigator)

You can select the location at which the data set will be stored:

- Temporary data storage (RAM disk), erased in the event of a power failure
- Permanent data storage (memory card) is retained if there is a power failure

Pay attention to consistency of the data to be backed up, see Consistent reading and writing of variables (semaphores) (Page 478).

When calling this function in short form, all parameters (including all optional parameters) must be specified.

The "Save variables" and "Restore variables" SCOUT functions can be used to retain data sets saved with _saveUnitDataSet, even if there is a change of version or data segments have been changed. For additional information, refer to the SIMOTION SCOUT Configuration Manual or the online help.

Declaration

```
_saveUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
          { path            : STRING,
            overwrite       : BOOL,
            nextCommand     : EnumNextCommandMode,
            dataScope       : EnumDeviceDataScope,
            kindOfData      : EnumDeviceKindOfData,
            }
        ): StructRetUnitDataSetCommand
```

Input parameter

unitName

Data type: STRING

Name of the unit (e.g. ST source file, MCC unit, LAD/FBD unit) whose unit variables will be saved.

The name must be indicated in lower case and inside single quotation marks (e.g. 'st_unit1').

If '_device' is specified, the global device variables will be saved.

id

Data type: UDINT

Number of the data set under which the variable values are saved (maximum of 1_000_000 data sets per unit).

storageType

Data type: EnumDeviceStorageType

Location at which the data is saved.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE,
        // Temporary data storage (RAM disk),
        // is deleted if there is a power failure
    PERMANENT_STORAGE,
        // Permanent data storage (memory card),
        //  is retained if there is a power failure
    USER_STORAGE )
        // with path specification
        // (only default permissible)
END_TYPE
```

path (optional)

Data type: STRING

Default: ' ' (empty string)

Destination path, if storageType = USER_STORAGE.

Only the default value may be specified. Otherwise, an error message is output.

overwrite (optional)

Data type: BOOL

Default: FALSE

If TRUE, existing data set is overwritten.

nextCommand (optional)

Data type: EnumNextCommandMode

Default: IMMEDIATELY

Advance to next command

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY,
        // Immediately
    WHEN_COMMAND_DONE )
        // After completion or abort of the command
END_TYPE
```

dataScope (optional)

Data type: EnumDeviceDataScope
Default _INTERFACE

```
TYPE EnumDeviceDataScope : (
    _INTERFACE,
        // Interface section
    _IMPLEMENTATION,
        // Implementation section
    _INTERFACE_AND_IMPLEMENTATION )
        // Interface and implementation section
END_TYPE
```

If unitName = '_device' is specified, only the values _INTERFACE or _INTERFACE_AND_IMPLEMENTATION are permissible for dataScope.

kindOfData (optional)

Data type: EnumDeviceKindOfData
Default NO_RETAIN_GLOBAL

Specification of whether non-retentive or retentive global variables will be saved.

```
TYPE EnumDeviceKindOfData : (
    NO_RETAIN_GLOBAL,
        // Non-retentive variables
    _RETAIN,
        // Retentive variables
    ALL_GLOBAL )
        // Retentive and non-retentive variables
END_TYPE
```

Return value

Data type: StructRetUnitDataSetCommand

The return value is a structure of data type StructRetUnitDataSetCommand. It comprises the following:

- A functionResult : EnumDeviceUnitDataSetCommand component.
  This supplies information on errors and the current state.
- A handle : UDINT component.
  This provides the possibility, by means of the _getStateOfUnitDataSetCommand (Page 442) function of checking the current state of a data backup function (especially useful with step enabling condition IMMEDIATELY).

Table 9-16 Data type return value

```
TYPE
  EnumDeviceUnitDataSetCommand: (
    DONE,       // Execution or start successful
    ACTIVE,                 // Being processed
    INTERNEL_ERROR,         // Internal error
            // (please call the hotline)
    COMMAND_FAILED,         // Command cannot be executed
    NO_COMMAND_BUFFER_AVAILABLE,  // Command buffer full
    COMMAND_NOT_FOUND,      // Command (handle) not found
    DATASET_ID_NOT_VALID,   // Data set number is invalid
    READ_ERROR,             // Data read error
                            // (defective storage medium)
    NO_STORAGE_AVAILABLE,   // No memory space available
    ACCESS_DENIED,          // Access denied
                            // (missing write/read rights)
    DATASET_ALREADY_EXISTS, // File already exists
    DATASET_NOT_FOUND,      // Data set not found
    UNIT_NOT_FOUND,         // Unit not found
            // (e.g. ST source file, MCC unit) not available
    VERSION_MISSMATCH,      // Incorrect version ID
            // of the data area to be imported
    DATA_INCOMPLETE,    // Incomplete import of the data
    SYMBOL_INFORMATION_NOT_AVAILABLE,
            // Symbol information not available
            //(Activate the OPC-XML export at the program source)
    DATA_MISMATCH,      // Data area to be imported
                        // is not contained in the data set
    DATA_INCOMPATIBLE )
            // Data set contains additional data segments or
            // selected data segments are missing in the data set

  StructRetUnitDataSetCommand  : STRUCT
    functionResult : EnumDeviceUnitDataSetCommand;
    handle : UDINT;
  END_STRUCT;
END_TYPE
```

For information about data backup, see "Data backup and data initialization from user program" (Page 481)

9.14.3 _loadUnitDataSet function

Description

The values of the following variables are loaded from a binary data set saved with _saveUnitDataSet (Page 428):

- Non-retentive or retentive global unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit, LAD/FBD unit)
- Non-retentive or retentive global device variables

The data set can only be loaded if the version codes of all the data segments to be loaded (e.g. non-retentive and retentive variables of the interface section of the unit) have remained unchanged since the data set was saved. For data segments and their version ID, see Section "Version ID of global variables and their initialization during download" in the Programming Manuals, e.g. SIMOTION ST.

A subset of the data segments backed up in the data set can be loaded (e.g. if the version codes of some data segments have changed).

When calling this function in short form, all parameters (including all optional parameters) must be specified.

The "Save variables" and "Restore variables" SCOUT functions can be used to retain data sets saved with _saveUnitDataSet, even if there is a change of version or data segments have been changed. For additional information, refer to the SIMOTION SCOUT Configuration Manual or the online help.

Declaration

```
_loadUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
          { path            : STRING,
            nextCommand     : EnumNextCommandMode,
            dataScope       : EnumDeviceDataScope,
            kindOfData      : EnumDeviceKindOfData,
            }
        ): StructRetUnitDataSetCommand
```

Input parameter

unitName

Data type: STRING

Name of the unit (e.g. ST source file, MCC unit), whose unit variables will be loaded.

The name must be indicated in lower case and inside single quotation marks (e.g. 'st_unit1').

If '_device' is specified, the global device variables will be loaded.

id

Data type: UDINT

Number of the data set from which the variable values are loaded (maximum of 1_000_000 data sets per unit).

storageType

Data type: EnumDeviceStorageType

Location at which the data is saved.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE,
        // Temporary data storage (RAM disk),
        // is deleted if there is a power failure
    PERMANENT_STORAGE,
        // Permanent data storage (memory card),
        // is retained if there is a power failure
    USER_STORAGE )
        // with path specification,
        // (only default permissible)
END_TYPE
```

path (optional)

Data type: STRING

Default: '' (empty string)

Destination path, if storageType = USER_STORAGE.

Only the default value may be specified. Otherwise, an error message is output.

nextCommand (optional)

Data type: EnumNextCommandMode

Default: IMMEDIATELY

Advance to next command

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY,
        // Immediately
    WHEN_COMMAND_DONE )
        // After completion or abort of the command
END_TYPE
```

dataScope (optional)

Data type: EnumDeviceDataScope

Default _INTERFACE

Specification of the section whose unit variables are to be saved.

```
TYPE EnumDeviceDataScope : (
    _INTERFACE,
        // Interface section
    _IMPLEMENTATION,
        // Implementation section
    _INTERFACE_AND_IMPLEMENTATION )
        // Interface and implementation section
END_TYPE
```

If unitName = '_device' is specified, only the values _INTERFACE or _INTERFACE_AND_IMPLEMENTATION are permissible for dataScope.

kindOfData (optional)

Data type: EnumDeviceKindOfData

Default NO_RETAIN_GLOBAL

Specification of whether non-retentive or retentive global variables will be loaded.

```
TYPE EnumDeviceKindOfData : (
    NO_RETAIN_GLOBAL,
        // Non-retentive variables
    _RETAIN,
        // Retentive variables
    ALL_GLOBAL )
        // Retentive and non-retentive variables
END_TYPE
```

If retentive and non-retentive variables are stored in the saved data set, it is possible to load retentive or non-retentive variables selectively.

Return value

Data type: StructRetUnitDataSetCommand

The return value is a structure of data type StructRetUnitDataSetCommand. It comprises the following:

- A *functionResult* : EnumDeviceUnitDataSetCommand. component.
  This supplies information on errors and the current state.
- A *handle* : UDINT component.
  This provides the possibility, by means of the _getStateOfUnitDataSetCommand (Page 442) function of checking the current state of a data backup function (especially useful with step enabling condition IMMEDIATELY).

For further information on data types StructRetUnitDataSetCommand and EnumDeviceUnit-DataSetCommand, see Section **Return value** of the _saveUnitDataSet (Page 431) function.

For information about data backup, see "Data backup and data initialization from user program" (Page 481)

9.14.4 _exportUnitDataSet function

Description

The values of the following variables are exported to a data record file:

- Non-retentive global unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit or LAD/FBD unit)
- Retentive global unit variables of the interface or the implementation section of a unit

You can select the location at which the data set will be stored:

- Temporary data storage (RAM disk), erased in the event of a power failure
- Permanent data storage (memory card) is retained if there is a power failure

The data record files are each stored under "\USER\SIMOTION\USER_DIR\UPP\UNITDS".

Non-retentive or retentive global device variables can only be backed up with the _saveUnitDataSet (Page 428) function.

Make sure the symbol information of the unit variables in the SIMOTION device is available for the relevant unit. Therefore, activate the **Permit OPC-XML** option in the local settings of the compiler at the program source, see Section "Local settings of the compiler" in the Programming Manuals, e.g. SIMOTION ST.

Pay attention to consistency of the data to be exported, see Consistent reading and writing of variables (semaphores) (Page 478).

The exported data set can also be imported with the _importUnitDataSet (Page 437) function if the version ID of the data area (e.g. retentive variables of the interface section of the unit) has changed (e.g. by a change to the data structure).

When calling this function in short form, all parameters (including all optional parameters) must be specified.

Declaration

```
_exportUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
          { path            : STRING,
            overwrite       : BOOL
            nextCommand     : EnumNextCommandMode,
            dataScope       : EnumDeviceDataScope,
            kindOfData      : EnumDeviceKindOfData,
            }
        ): StructRetUnitDataSetCommand
```

Input parameter

unitName

Data type: STRING

Name of the unit (e.g. ST source file, MCC unit), whose unit variables will be exported.

The name must be indicated in lower case and inside single quotation marks (e.g. 'st_unit1').

Specifying '_device' (for exporting global device variables) is not permissible here (only possible with _saveUnitDataSet (Page 428)).

id

Data type: UDINT

Number of the data set under which the variable values are saved (maximum of 1_000_000 data sets per unit).

storageType

Data type: EnumDeviceStorageType

Location at which the data is saved.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE,
        // Temporary data storage (RAM disk),
        // is deleted if there is a power failure
    PERMANENT_STORAGE,
        // Permanent data storage (memory card),
        //  is retained if there is a power failure
    USER_STORAGE )
        // with path specification
        // (only default permissible)
END_TYPE
```

path (optional)

Data type: STRING

Default: ' ' (empty string)

Destination path, if storageType = USER_STORAGE.

Only the default value may be specified. Otherwise, an error message is output.

The resulting files from _exportUnitDataSet() are intended for use in combination with _importUnitDataSet() only.

overwrite (optional)

Data type: BOOL

Default: FALSE

If TRUE, existing data set is overwritten.

nextCommand (optional)

Data type: EnumNextCommandMode

Default: IMMEDIATELY

Advance to next command

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY,
        // Immediately
    WHEN_COMMAND_DONE )
        // After completion or abort of the command
END_TYPE
```

dataScope (optional)

Data type: EnumDeviceDataScope

Default _INTERFACE

Specification of the section whose unit variables will be exported.

```
TYPE EnumDeviceDataScope : (
    _INTERFACE,
        // Interface section
    _IMPLEMENTATION,
        // Implementation section
    _INTERFACE_AND_IMPLEMENTATION )
        // Interface and implementation section
END_TYPE
```

kindOfData (optional)

Data type: EnumDeviceKindOfData

Default NO_RETAIN_GLOBAL

Specification of whether non-retentive or retentive global variables will be exported.

```
TYPE EnumDeviceKindOfData : (
    NO_RETAIN_GLOBAL,
        // Non-retentive variables
    _RETAIN,
        // Retentive variables
    ALL_GLOBAL )
        // Retentive and non-retentive variables
END_TYPE
```

Return value

Data type: StructRetUnitDataSetCommand

The return value is a structure of data type StructRetUnitDataSetCommand. It comprises the following:

- A *functionResult* : EnumDeviceUnitDataSetCommand component.
  This supplies information on errors and the current state.
- A *handle* : UDINT component.
  This provides the possibility, by means of the _getStateOfUnitDataSetCommand (Page 442) function of checking the current state of a data backup function (especially useful with step enabling condition IMMEDIATELY).

For further information on data types StructRetUnitDataSetCommand and EnumDeviceUnit-DataSetCommand, see Section **Return value** of the _saveUnitDataSet (Page 431) function.

For information about data backup, see "Data backup and data initialization from user program" (Page 481)

9.14.5 _importUnitDataSet function

Description

The values of the following variables are imported from a data set that was exported with _exportUnitDataSet (Page 434).

- Non-retentive global unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit or LAD/FBD unit)
- Retentive global unit variables of the interface or the implementation section of a unit

Make sure the symbol information of the unit variables in the SIMOTION device is available for the relevant unit. Therefore, activate the **Permit OPC-XML** option in the local settings of the compiler at the program source, see Section "Local settings of the compiler" in the Programming Manuals, e.g. SIMOTION ST.

The data set can also be imported if the version code of a data segment to be loaded has changed since the data set was saved (e.g. non-retentive variables of the interface section of the unit):

- Variables that no longer exist are ignored.
- The value of added variables remains unchanged.

- In the case of a variable with a changed data type, the value is transferred if it can be converted to the new data type; otherwise the value of the variable is retained.
- The returned value of the function is DATA_INCOMPLETE.

To avoid unwanted values in variables, you can initialize the relevant data segments before importing the data set with the _resetUnitData function (see List Manual for the system functions of the SIMOTION devices).

For data segments and their version ID, see Section "Version ID of global variables and their initialization during download" in the Programming Manuals, e.g. SIMOTION ST.

A subset of the data segments exported in the data set can be imported.

When calling this function in short form, all parameters (including all optional parameters) must be specified.

Declaration

```
_importUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
          { path            : STRING,
            nextCommand     : EnumNextCommandMode,
            dataScope       : EnumDeviceDataScope,
            kindOfData      : EnumDeviceKindOfData,
            }
        ): StructRetUnitDataSetCommand
```

Input parameter

unitName

Data type: STRING

Name of the unit (e.g. ST source file, MCC unit), whose unit variables will be imported.

The name must be indicated in lower case and inside single quotation marks (e.g. 'st_unit1').

Specifying '_device' (for importing global device variables) is not permissible here (only possible with _loadUnitDataSet (Page 431)).

id

Data type: UDINT

Number of the data set under which the variable values are saved (maximum of 1_000_000 data sets per unit).

storageType

Data type: EnumDeviceStorageType

Location at which the data is saved.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE,
        // Temporary data storage (RAM disk),
        // is deleted if there is a power failure
    PERMANENT_STORAGE,
        // Permanent data storage (memory card),
        //  is retained if there is a power failure
    USER_STORAGE )
        // with path specification
        // (only default permissible)
END_TYPE
```

path (optional)

Data type: STRING

Default: ' ' (empty string)

Destination path, if storageType = USER_STORAGE.

Only the default value may be specified. Otherwise, an error message is output.

overwrite (optional)

Data type: BOOL

Default: FALSE

If TRUE, existing data set is overwritten.

nextCommand (optional)

Data type: EnumNextCommandMode

Default: IMMEDIATELY

Advance to next command

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY,
        // Immediately
    WHEN_COMMAND_DONE )
        // After completion or abort of the command
END_TYPE
```

dataScope (optional)

Data type: EnumDeviceDataScope

Default _INTERFACE

Specification of the section whose unit variables will be imported.

```
TYPE EnumDeviceDataScope : (
    _INTERFACE,
        // Interface section
    _IMPLEMENTATION,
        // Implementation section
    _INTERFACE_AND_IMPLEMENTATION )
        // Interface and implementation section
END_TYPE
```

kindOfData (optional)

Data type: EnumDeviceKindOfData

Default NO_RETAIN_GLOBAL

Specification of whether non-retentive or retentive global variables will be imported.

```
TYPE EnumDeviceKindOfData : (
    NO_RETAIN_GLOBAL,
        // Non-retentive variables
    _RETAIN,
        // Retentive variables
    ALL_GLOBAL )
        // Retentive and non-retentive variables
END_TYPE
```

If retentive and non-retentive variables are stored in the exported data set, it is possible to import retentive or non-retentive variables selectively.

Return value

Data type: StructRetUnitDataSetCommand

The return value is a structure of data type StructRetUnitDataSetCommand. It comprises the following:

- A *functionResult* : EnumDeviceUnitDataSetCommand component.
  This supplies information on errors and the current state.
- A *handle* : UDINT. component.
  This provides the possibility, by means of the _getStateOfUnitDataSetCommand (Page 442) function of checking the current state of a data backup function (especially useful with step enabling condition IMMEDIATELY).

For further information on data types StructRetUnitDataSetCommand and EnumDeviceUnit-DataSetCommand, see Section **Return value** of the _saveUnitDataSet (Page 431) function.

For information about data backup, see "Data backup and data initialization from user program" (Page 481)

9.14.6 _deleteUnitDataSet function

Description

A single data set containing the stored values of the following variables is deleted:

- Backed-up non-retentive or retentive unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit)
- Backed-up non-retentive or retentive global device variables.
- Exported non-retentive or retentive unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit)

When calling this function in short form, all parameters (including all optional parameters) must be specified.

Declaration

```
_deleteUnitDataSet (
            unitName       : STRING,
            id             : UDINT,
            storageType    : EnumDeviceStorageType,
          { path           : STRING,
            nextCommand    : EnumNextCommandMode,
            }
        ): StructRetUnitDataSetCommand
```

Input parameter

unitName

Data type: STRING

Name of the unit (e.g. ST source file, MCC unit); the name must be written in lower case and placed inside single quotation marks (e.g. 'st_unit1').

If '_device' is specified, a data set for global device variables will be deleted.

id

Data type: UDINT

Number of the data set to be deleted (maximum of 1_000_000 data sets per unit).

storageType

Data type: EnumDeviceStorageType

Location from which the data set is to be deleted.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE,
        // Temporary data storage (RAM disk),
        // is deleted if there is a power failure
    PERMANENT_STORAGE,
        // Permanent data storage (memory card),
        //  is retained if there is a power failure
    USER_STORAGE )
        // with path specification
        // (only default permissible)
END_TYPE
```

path (optional)

Data type: STRING

Default: ' ' (empty string)

Destination path, if storageType = USER_STORAGE.

Only the default value may be specified. Otherwise, an error message is output.

nextCommand (optional)

Data type: EnumNextCommandMode
Default: IMMEDIATELY
Advance to next command

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY,
        // Immediately
    WHEN_COMMAND_DONE )
        // After completion or abort of the command
END_TYPE
```

Return value

Data type: StructRetUnitDataSetCommand

The return value is a structure of data type StructRetUnitDataSetCommand. It comprises the following:

- A *functionResult* : EnumDeviceUnitDataSetCommand component.
  This supplies information on errors and the current state.
- A *handle* : UDINT component.
  This provides the possibility, by means of the _getStateOfUnitDataSetCommand (Page 442) function of checking the current state of a data backup function (especially useful with step enabling condition IMMEDIATELY).

For further information on data types StructRetUnitDataSetCommand and EnumDeviceUnit-DataSetCommand, see Section Return value of the _saveUnitDataSet (Page 431) function.

For information about data backup, see "Data backup and data initialization from user program" (Page 481)

See also

_deleteAllUnitDataSets function (Page 445)

9.14.7 _getStateOfUnitDataSetCommand function

Description

It returns the state of the functions for data backup.

Declaration

```
_getStateOfUnitDataSetCommand (
            handle  : UDINT
        ) : EnumDeviceUnitDataSetCommand
```

Input parameter

handle

Data type: UDINT

Handle of the data backup function for which the state is to be checked. You received this handle as a component of the return value of the data backup function.

Return value

Data type: EnumDeviceUnitDataSetCommand

This supplies information on errors and the current state.

For further information on the EnumDeviceUnitDataSetCommand data type, see Section **Return value** of the _saveUnitDataSet (Page 431) function.

For information about data backup, see "Data backup and data initialization from user program" (Page 481).

9.14.8 _checkExistingUnitDataSet function

Description

A check is performed to determine whether the specified data set containing the stored values of the following variables is available on the storage medium:

- Backed-up non-retentive or retentive unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit)
- Backed-up non-retentive or retentive global device variables.
- Exported non-retentive or retentive unit variables of the interface section of a unit (e.g. ST source file, MCC unit)

When calling this function in short form, all parameters (including all optional parameters) must be specified.

Declaration

```
_checkExistingUnitDataSet (
            unitName       : STRING,
            id             : UDINT,
            storageType    : EnumDeviceStorageType,
          { path           : STRING,
            nextCommand    : EnumNextCommandMode,
            }
        ) : StructRetUnitDataSetCommand
```

Input parameter

unitName

Data type: STRING

Name of the unit (e.g. ST source file, MCC unit); the name must be written in lower case and placed inside single quotation marks (e.g. 'st_unit1').

If '_device' is specified, a data set for global device variables will be checked (possible as of Version 3.2 of the SIMOTION Kernel).

id

Data type: UDINT

Number of the data set (max. 1_000_000 data sets per unit).

storageType

Data type: EnumDeviceStorageType

Location at which the data is stored.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE,
        // Temporary data storage (RAM disk),
        // is deleted if there is a power failure
    PERMANENT_STORAGE,
        // Permanent data storage (memory card),
        //  is retained if there is a power failure
    USER_STORAGE )
        // with path specification
        // (only default permissible)
END_TYPE
```

path (optional)

Data type: STRING

Default: ' ' (empty string)

Destination path, if storageType = USER_STORAGE.

Only the default value may be specified. Otherwise, an error message is output.

nextCommand (optional)

Data type: EnumNextCommandMode

Default: IMMEDIATELY

Advance to next command

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY,
        // Immediately
    WHEN_COMMAND_DONE )
        // After completion or abort of the command
END_TYPE
```

Return value

Data type: StructRetUnitDataSetCommand

The return value is a structure of data type StructRetUnitDataSetCommand. It comprises the following:

- A *functionResult* : EnumDeviceUnitDataSetCommand component.
  This supplies information on errors and the current state.
- A *handle* : UDINT component.
  This provides the possibility, by means of the _getStateOfUnitDataSetCommand (Page 442) function of checking the current state of a data backup function (especially useful with step enabling condition IMMEDIATELY).

For further information on data types StructRetUnitDataSetCommand and EnumDeviceUnit-DataSetCommand, see Section Return value of the _saveUnitDataSet (Page 431) function.

For information about data backup, see "Data backup and data initialization from user program" (Page 481).

9.14.9 _deleteAllUnitDataSets function

Description

All data sets containing stored values for the following variables are deleted.

- Backed-up non-retentive or retentive unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit)
- Backed-up non-retentive or retentive global device variables.
- Exported non-retentive or retentive unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit)

When calling this function in short form, all parameters (including all optional parameters) must be specified.

Declaration

```
_deleteAllUnitDataSets (
            unitName       : STRING,
            storageType    : EnumDeviceStorageType,
          { path           : STRING,
            nextCommand    : EnumNextCommandMode,
            }
        ) : StructRetUnitDataSetCommand
```

Input parameter

unitName

Data type: STRING

Name of the unit (e.g. ST source file, MCC unit); the name must be written in lower case and placed inside single quotation marks (e.g. 'st_unit1').

If '_device' is specified, all data sets for global device variables will be deleted (possible as of Version 3.2 of the SIMOTION Kernel).

storageType

Data type: EnumDeviceStorageType

Location from which the data set is to be deleted.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE,
        // Temporary data storage (RAM disk),
        // is deleted if there is a power failure
    PERMANENT_STORAGE,
        // Permanent data storage (memory card),
        //  is retained if there is a power failure
    USER_STORAGE )
        // with path specification
        // (only default permissible)
END_TYPE
```

path (optional)

Data type: STRING

Default: ' ' (empty string)

Destination path, if storageType = USER_STORAGE.

Only the default value may be specified. Otherwise, an error message is output.

nextCommand (optional)

Data type: EnumNextCommandMode

Default: IMMEDIATELY

Advance to next command

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY,
        // Immediately
    WHEN_COMMAND_DONE )
        // After completion or abort of the command
END_TYPE
```

Return value

Data type: StructRetUnitDataSetCommand

The return value is a structure of data type StructRetUnitDataSetCommand. It comprises the following:

- A *functionResult* : EnumDeviceUnitDataSetCommand component.
  This supplies information on errors and the current state.
- A *handle* : UDINT component.
  This provides the possibility, by means of the _getStateOfUnitDataSetCommand (Page 442) function of checking the current state of a data backup function (especially useful with step enabling condition IMMEDIATELY).

For further information on data types StructRetUnitDataSetCommand and EnumDeviceUnit-DataSetCommand, see Section **Return value** of the _saveUnitDataSet (Page 428) function.

For information about data backup, see "Data backup and data initialization from user program".

9.15 Functions for commandId

9.15.1 CommandID overview

Description

A CommandId is transferred every time a command is issued. This is saved on the command and represents a reference to the issued command.

A command is thus uniquely identifiable and traceable using the CommandId.

Note

If the CommandId is not assigned for the command call, the default value of the parameter takes effect with (0.0).

Further information

Further information can be found at Command execution diagnostics (Page 137), Transition and step enabling conditions (Page 131) and Using the commandId parameter correctly (Page 592).

9.15.2 _getCommandId function

Description

This function supplies a project-wide unique commandId, which can be used for explicit identification of commands.

A commandId is always produced, there is no error feedback.

Declaration

```
_getCommandId ( ) : CommandIdType
```

Input parameters

None

Return value

Data type: CommandIdType

Project-wide unique CommandId for tracking the command status

```
TYPE
 CommandIdType : STRUCT
    SystemId_low        : UDINT;    // Lower-order part
    SystemId_high       : UDINT;    // Higher-order part
 END_STRUCT
END_TYPE
```

See also

Using the commandId parameter correctly (Page 592)

9.15.3 _getSyncCommandId function

Description

This function supplies the user with a project-wide unique syncCommandId. This ID can be transferred to system functions BEGIN_SYNC and _startSyncCommands (see Parameter Manuals for SIMOTION devices) to start motion sequences synchronously.

A syncCommandId is always produced, there is no error feedback.

Declaration

```
_getSyncCommandId ( ) : CommandIdType
```

Input parameter

None

Return value

Data type: CommandIdType
Project-wide unique syncCommandId for tracking the command status.

```
TYPE
 CommandIdType : STRUCT
    SystemId_low        : UDINT;    // Lower-order part
    SystemId_high       : UDINT;    // Higher-order part
 END_STRUCT
END_TYPE
```

See also

Using the commandId parameter correctly (Page 592)

9.16 Defining the waiting time

9.16.1 _waitTime function

Description

This function interrupts the task triggering this function until the time specified in the call has expired.

Note

The function should be used in MotionTasks only; using it in cyclic tasks may lead to time monitoring errors!

- With SynchronousTasks: You can configure whether the time watchdog is suspended. Time monitoring is active by default.
  With IPOsynchronousTask, additionally take the following into account: UserInterruptTasks will no longer be started by their triggering event!
- With other cyclic tasks (BackgroundTask, TimerInterruptTasks): Time monitoring is always active.

In cyclic tasks, use the system function blocks Timers (Page 535) to implement wait times.

The _waitTime function is always executable, its return value = 0.

For information on how you can make a task wait for a certain time, see Making tasks wait a defined period (Page 337).

Declaration

```
_waitTime (
        timeValue    : TIME    // Wait time
        ) : DINT               // Always = 0
```

Input parameter

timeValue

Data type: TIME

Indicates the time interval during which task processing is interrupted.

Return value

Data type: DINT

Is always 0.

See also

Time allocation in the round robin execution level (Page 273)
Wait times in cyclic tasks (Page 592)

9.17 Device-specific functions

9.17.1 _getDeviceId function

Description

The function reads the hardware ID of the SIMOTION device from its hardware information block. You specify the type of ID to be read as input parameter when calling the function.

Declaration

```
_getDeviceId (
              idType : EnumDeviceIdType
             ) : StructRetGetDeviceId
```

Input parameter

idType

Data type: EnumDeviceIdType

Specification of the ID to be read

```
TYPE EnumDeviceIdType : (
    SERIAL_NUMBER ,    // CPU serial number
    HW_TYPE ,          // Module type
    SPECIFIC_NUMBER ,  // Special OEM number
    ORDER_ID )         // Order number of the module
END_TYPE
```

Return value

Data type: StructRetGetDeviceId

The return value is a structure of data type StructRetGetDeviceId. It comprises the following:

- A *functionResult* component: DINT.
  This provides information on errors.
- An *id* component: STRING[254]
  This contains the read hardware ID of the memory card.

```
TYPE
  StructRetGetDeviceId : STRUCT
    functionResult : UDINT; // Error status
        // 16#00000000 : No error
        // 16#FFFF80C3 : Information not available
        // 16#FFFF8090 : Incorrect transfer parameter
        // 16#FFFF8099 : Internal error
    id : STRING[254];       // Read-out hardware ID
  END_STRUCT;
END_TYPE
```

9.17.2 _getMemoryCardId function

Description

The function reads the hardware ID of a memory card from its hardware information block. You specify the type of ID to be read as input parameter when calling the function (at present only serial number possible).

Declaration

```
_getMemoryCardId (
                  idType : EnumMemoryCardIdType
                 ) : StructRetGetMemoryCardId
```

Input parameter

idType

Data type: EnumMemoryCardIdType

Specification of the ID to be read

```
TYPE EnumMemoryCardIdType : (
    MEMORY_CARD_SERIAL_NUMBER ) // Memory card serial number
END_TYPE
```

Return value

Data type: StructRetGetMemoryCardId

The return value is a structure of data type StructRetGetMemoryCardId. It comprises the following:

- A *functionResult* component: DINT.
  This provides information on errors.
- An *id* component: STRING[254]
  This contains the read hardware ID of the memory card.

```
TYPE
  StructRetGetMemoryCardId : STRUCT
    functionResult : UDINT; // Error status
        // 16#0000_0000 : No error
        // 16#FFFF_FFFD : Internal error
        // 16#FFFF_FFF8 : Incorrect parameter
    id : STRING[254];       // Read-out hardware ID
  END_STRUCT;
END_TYPE
```

Note

Any leading "S" in the memory card serial number will be ignored and not returned in the return value. For determining the license key in the Web License Manager, it is irrelevant whether or not the serial number is specified with a leading "S".

9.17.3 _setDeviceErrorLED function

Description

The function sets the **Underlicensing of technology/option objects** error on the SIMOTION device. The corresponding LED flashes in red on the SIMOTION device (see Manual of the SIMOTION device).

Declaration

```
_setDeviceErrorLED ( ) : DINT
```

Input parameter

None

Return value

Data type:	DINT
16#0000_0000	No error
16#FFFF_FFFD	Internal error

9.17.4 _setDriveObjectSTW function

Description

The _setDriveObjectSTW system function can be used to set individual bits of the CU_STW control word for drive object 1 (DO1) of the drive unit. This control word is part of the manufacturer-specific PROFIdrive telegrams 390 ff (telegrams 39x) for the control unit of the drive unit.

No bits can be set that have been reserved for the communication between the SIMOTION device and the drive unit or for other purposes.

This function is available in the SIMOTION device as of version V4.1 SP2.

Declaration

```
_setDriveObjectSTW (
            logAddress  : DINT;
            STW1BitMask : UINT;
            STW1BitSet  : UINT
           ) : DINT
```

Input parameter

logAddress

Data type DINT

Logical base address of the send telegram for the drive object.

STW1BitMask

Data type UNIT

Bit mask for selecting the affected bits in the CU_STW control word of the send telegram.

Bits that are operated autonomously in the connection to the drive object of SIMOTION (e.g. for the synchronization, fault handling, etc.) cannot be accessed by this function (see the table below for the assignment and reservation of the bits in the CU_STW).

Bits that can be accessed via the function:

- If time synchronization for Sinamics is configured: 16#007C = 0111 1100
- If no time synchronization is configured for Sinamics: 16#007E = 0111 1110

STW1BitSet

Data type UNIT

Bit-coded value for writing in the CU_STW control word of the send telegram. Only those bits selected via STW1BitMask are written.

Return value

Data type: DINT

The return value informs the user about the result of the call.

16#0000_0000	Request successfully completed • Bits have been written in the control word • Bits in STW1 were already in the defined state
16#FFFF_8090	Job aborted Logical address is not available
16#FFFF_8091	Job aborted Addressed drive object does not support the function
16#FFFF_8093	Job aborted STW1BitMask contains bits blocked for the function
16#FFFF_809F	Job aborted Internal error, function cannot be executed, e.g. function called on SIMOTION devices of version V4.1 or V4.1 SP1

Bit assignment and reservation in CU_STW of the 39x telegrams

Bit	Name	Meaning	Access right
12-15	MLZ	Dyn. master sign-of-life	SIMOTION FW
11	-	Reserved - system	Reserved
10	DAG	Demand from AG	SIMOTION FW
9	-	Reserved - system	Reserved
8	-	Reserved - system	Reserved
7	FACK	Fault acknowledge	_resetDriveObjectFault
6	-		_setDriveObjectSTW
5	-		_setDriveObjectSTW
4	-		_setDriveObjectSTW
3	-		_setDriveObjectSTW
2	-		_setDriveObjectSTW

Bit	Name	Meaning	Access right
1	PING	Start pulse for time synchronization	_setDriveObjectSTW / SIMOTION-FW*
0	SYN	Dyn. synchronization flag for system time cycle	SIMOTION FW

* Bit 1 can be written using _setDriveObjectSTW if no time synchronization for SINAMICS is configured.

9.18 Determine the memory size of a variable or of a data type

9.18.1 _sizeOf function

Description

The function returns the memory size in bytes required for a variable or data type:

- By default, the return value is determined at the time of compilation. The function can then be used in constant expressions (e.g. during initialization).
- Exception - only as of Version 4.2 of the SIMOTION kernel:
  In the case of an ARRAY with a dynamic length (i.e. the index limits are not declared), the return value is determined during runtime.

Declaration

```
_sizeOf (
         in  : ANY  // Identifier of the data type or
                    // of the variables
         )   : DINT
```

Input parameter

in

Data type: ANY

Identifier of the variable or data type, whose size is to be determined.

Return value

Data type: DINT

Required memory size in bytes.

The memory size is specified taking account of the natural layout, i.e. in accordance with the assignment possibilities of the data types in the memory. Therefore, the effective size is determined, which is required for the use of the data type in an ARRAY.

The actual required size may be less.

Example

```
TYPE
    a_type  : STRUCT
        a   : LREAL;    // 8 bytes
        b   : BOOL;     // 1 byte
    END_STRUCT;
END_TYPE
//...
x := _sizeOf (in := a_type);    // Returns value 16
```

9.18.2 _firstIndexOf function

Description

This function returns the lower index limit for an ARRAY (variable or data type):

- In the case of an ARRAY with a defined length (i.e. the index limits are declared), the return value is determined at the time of compilation. The function can then be used in constant expressions (e.g. during initialization).
- Only as of Version V4.2 of the SIMOTION kernel:
  In the case of an ARRAY with a dynamic length (i.e. the index limits are not declared), the return value is determined during runtime.

Note

The function must not be used for variables of the STRING data type or directly on this data type. The function then returns the value 1.

Declaration

```
_firstIndexOf (
        in  : ARRAY [..] OF ANY  // Identifier of the ARRAY
                                 // (Data type or variable)
        )   : DINT
```

Input parameter

in

Data type: ARRAY [..] OF ANY
Identifier for the ARRAY (variable or data type) whose lower index limit is to be determined

Return value

Data type: DINT
Lower index limit

Example

```
TYPE
    array_1  : ARRAY [5..29] OF DINT;
    string_1 : STRING[80];
END_TYPE
// ...
x := _firstIndexOf (in := array_1);      // Returns value 5
y := _firstIndexOf (in := string_1);     // Returns value 1
```

9.18.3 _lastIndexOf function

Description

This function returns the upper index limit for an ARRAY (variable or data type):

- In the case of an ARRAY with a defined length (i.e. the index limits are declared), the return value is determined at the time of compilation. The function can then be used in constant expressions (e.g. during initialization).
- Only as of Version V4.2 of the SIMOTION kernel:
  In the case of an ARRAY with a dynamic length (i.e. the index limits are not declared), the return value is determined during runtime.

Note

The function must not be used for variables of the STRING data type or directly on this data type. The function then returns the last index permitted for indexed access (**not** the currently assigned length of the string).

Declaration

```
_lastIndexOf (
        in  : ARRAY [..] OF ANY  // Identifier of the ARRAY
                                 // (Data type or variable)
        )   : DINT
```

Input parameter

in

Data type: ARRAY [..] OF ANY

Identifier for the ARRAY (variable or data type) whose upper index limit is to be determined

Return value

Data type: DINT

Upper index limit.

Example

```
TYPE
    array_1  : ARRAY [5..29] OF DINT;
    string_1 : STRING[80];
END_TYPE
// ...
x := _lastIndexOf (in := array_1);      // Returns value 29
y := _lastIndexOf (in := string_1);     // Returns value 80
```

9.18.4 _lengthIndexOf function

Description

This function returns the number of array elements for an ARRAY (variable or data type):

- In the case of an ARRAY with a defined length (i.e. the index limits are declared), the return value is determined at the time of compilation. The function can then be used in constant expressions (e.g. during initialization).
- Only as of Version V4.2 of the SIMOTION kernel:
  In the case of an ARRAY with a dynamic length (i.e. the index limits are not declared), the return value is determined during runtime.

Note

The function must not be used for variables of the STRING data type or directly on this data type. The function returns the usable length of the string (identical with the return value of the _lastIndexOf function.

Declaration

```
_lengthIndexOf (
        in  : ARRAY [..] OF ANY  // Identifier of the ARRAY
                                 // (Data type or variable)
        )   : DINT
```

Input parameter

in

Data type: ARRAY [..] OF ANY

Identifier of the ARRAY (variable or data type) for which the number of array elements is to be determined

Return value

Data type: DINT

Number of array elements

The following applies:
_lengthIndexOf (*array name*) := _lastIndexOf (*array name*) - _firstIndexOf (*array name*) +1

Example

```
TYPE
    array_1 : ARRAY [5..29] OF DINT;
END_TYPE
// ...
x := _lengthIndexOf (in := array_1);      // Returns value 25
```

9.18.5 LOWER_BOUND function

Description

This function returns the lower index limit for an ARRAY (variable or data type):

- In the case of an ARRAY with a defined length (i.e. the index limits are declared), the return value is determined at the time of compilation. The function can then be used in constant expressions (e.g. during initialization).
- Only as of Version V4.2 of the SIMOTION kernel:
  In the case of an ARRAY with a dynamic length (i.e. the index limits are not declared), the return value is determined during runtime.

To use this function, you must activate the "Permit language extensions IEC61131-3rd Edition" compiler option, see SIMOTION ST Programming and Operating Manual.

Note

The function must not be used for variables of the STRING data type or directly on this data type. The function then returns the value 1.

Declaration

```
LOWER_BOUND (
        arr   : ARRAY [..] OF ANY  // Identifier of the ARRAY
                                   // (Data type or variable)
        { dim : UDINT          // ARRAY dimension
        }                      // (Only constant value 1 permitted)
        )     : DINT
```

Input parameter

arr

Data type: ARRAY [..] OF ANY

Identifier for the ARRAY (variable or data type) whose lower index limit is to be determined

dim (optional)

Data type: UDINT

Default setting: 1

ARRAY dimension (only constant value 1 permitted)

It may be specified only when a variable was passed in *arr*.

Return value

Data type: DINT
Lower index limit

Example

```
TYPE
    array_1  : ARRAY [5..29] OF DINT;
    string_1 : STRING[80];
END_TYPE
// ...
x := LOWER_BOUND (arr := array_1);      // Returns value 5
y := LOWER_BOUND (arr := string_1);     // Returns value 1
```

9.18.6 UPPER_BOUND function

Description

This function returns the upper index limit for an ARRAY (variable or data type):

- In the case of an ARRAY with a defined length (i.e. the index limits are declared), the return value is determined at the time of compilation. The function can then be used in constant expressions (e.g. during initialization).
- Only as of Version V4.2 of the SIMOTION kernel:
  In the case of an ARRAY with a dynamic length (i.e. the index limits are not declared), the return value is determined during runtime.

To use this function, you must activate the "Permit language extensions IEC61131-3rd Edition" compiler option, see SIMOTION ST Programming and Operating Manual.

Note

The function must not be used for variables of the STRING data type or directly on this data type. The function then returns the last index permitted for indexed access (**not** the currently assigned length of the string).

Declaration

```
UPPER_BOUND (
        arr   : ARRAY [..] OF ANY  // Identifier of the ARRAY
                                   // (Data type or variable)
        { dim : UDINT          // ARRAY dimension
        }                      // (Only constant value 1 permitted)
        )     : DINT
```

Input parameter

arr

Data type: ARRAY [..] OF ANY

Identifier for the ARRAY (variable or data type) whose upper index limit is to be determined

dim (optional)

Data type: UDINT

Default setting: 1

ARRAY dimension (only constant value 1 permitted)

It may be specified only when a variable was passed in *arr*.

Return value

Data type: DINT

Upper index limit.

Example

```
TYPE
    array_1  : ARRAY [5..29] OF DINT;
    string_1 : STRING[80];
END_TYPE
// ...
x := UPPER_BOUND (arr := array_1);      // Returns value 29
y := UPPER_BOUND (arr := string_1);     // Returns value 80
```

9.19 Determination of the logical address for an I/O variable

9.19.1 Function _getLogicalAdressOfIoVariable

Description

The function determines the start address of an I/O variable for direct access or process image of the cyclical tasks.

This function is especially useful if a symbolic assignment of the I/O variables to the addresses (Page 95) is performed. Some functionalities, such as the evaluation of alarms or TSI, require the knowledge of the logical address for an I/O variable in advance.

When using the function, the standard rules for using I/O variables apply.

Note

Every change in the address list for I/O (e.g. changes to the I/O addresses, new I/O variables) requires a recompilation of all program sources which contain this function.

Declaration

```
_getLogicalAddressOfIoVariable (
                in : <IO_VARIABLE> // Identifier of an I/O variable
            ) : DINT
```

Input parameter

in

Data type: <IO_VARIABLE>

Identifier of an I/O variable for direct access or the process image of the cyclical tasks. The I/O variable must be defined in the address list of the detail view.

With arrays of I/O variables, the specification of a constant index is permitted.

Return value

Data type: DINT

Logical start address of the I/O variable. It is returned in the following value ranges:

- For I/O variables for inputs: 16#0000 .. 16#FFFF
- For I/O variables for outputs: 16#7000_0000 .. 16#7000_FFFF

9.20 Copy areas of arrays

9.20.1 _ArrayCopy function

Description

The function copies an area of an array (copy source) to any location of another array of the same base data type (copy target). This area is specified by:

- Specification of the area length to be copied (number of array elements to be copied)
- Optional specification of the area start to be copied within the copy source.
- Optional specification of the area start to be written within the copy target.

The base data type of the copy source and the copy target must be identical; the index specifications may, however, differ. For information about the base data type and the index specification: see SIMOTION ST Programming and Operating Manual.

The copy target may also be identical with the copy source; the correct copy sequence of the array elements is selected automatically.

Note

If the area to be copied or written lies completely or partially outside the array limits of the copy source or copy target, the following applies: The array limits determine the maximum number of copied array elements.

If the compiler already determines that no array elements are copied (return value = 0), a warning will be issued.

The function can be interrupted by other tasks. The consistency of the copy target so cannot be guaranteed.

Declaration

```
_ArrayCopy (
        Dest   : ARRAY [..] OF ANY  // Copy target
        Source : ARRAY [..] OF ANY  // Copy source
        Length : DINT    // Number of array elements to be copied
        { DestIndex   : DINT  // Start of the area to be written
          SourceIndex : DINT  // Start of the area to be copied
         }
         )     : DINT
```

In-out parameter

Dest

Data type: ARRAY [..] OF ANY

Copy target.

Array into which the area is to be copied.

The copy source and the copy target must be arrays of the same base data type, although they can have different index specifications. For information about the base data type and the index specification: see SIMOTION ST Programming and Operating Manual.

Source

Data type: ARRAY [..] OF ANY

Copy source.

Array from which the area is to be copied.

The copy source and the copy target must be arrays of the same base data type, although they can have different index specifications. For information about the base data type and the index specification: see SIMOTION ST Programming and Operating Manual.

Input parameter

Length

Data type: DINT

Length of the area to be copied (number of array elements to be copied)

DestIndex (optional)

Data type: DINT

Default setting: 0

Start of the area within the copy target into which the copied array elements are written (index of the copy target into which the first copied array element is written).

The value must lie within the declared array limits of the copy target.

SourceIndex (optional)

Data type: DINT

Default setting: 0

Start of the area to be copied within the copy source (index of the first copied array).

The value must lie within the declared array limits of the copy source.

Return value

Data type: DINT

Number of copied array elements

For a successful and error-free execution of the function, the return value represents the length of the area to be copied.

Example

```
VAR
    src_array  : ARRAY [6..36] OF DINT;
    dest_array : ARRAY [0..25] OF DINT;
    copy_length : DINT := 15;
END_VAR
// ...
ret := _ArrayCopy (
        Dest        := dest_array,
        Source      := src_array,
        Length      := copy_length,
        DestIndex   := 20,
        SourceIndex := 10);
IF ret = copy_length THEN
    ; // Copy successful
ELSE
    ; // Error handling
END_IF;
```

9.21 Display the intermediate results for "program status"

9.21.1 _trcVal function

Description

The function extends the display options for the "Program status" diagnostic function for the SIMOTION ST programming language

By default, the program status shows only the values for simple variables of elementary data types. The intermediate results of formulas can also be displayed with the _trcVal function. To do this, pass the expression to be displayed as actual parameter to the _trcVal function.

The "Program status" display window shows the return value for each _trcVal function call in the form: "`_TRCVAL = ...`".

Note

If the _trcVal function is used for variables that cannot be displayed in the "Program status" (e.g. no simple variable), the compiler issues an error message.

The function cannot be used for enumerations.

For information about the "Program status" diagnostic function, see SIMOTION ST Programming and Operating Manual.

Declaration

```
_trcVal (
        in  : ANY_ELEMENTARY
        )   : ANY_ELEMENTARY
```

Input parameter

in

Data type: ANY_ELEMENTARY

The expression to be displayed in the "Program status".

Return value

Data type: ANY_ELEMENTARY

The same data type as the *in* input parameter.

Example

In the following example, the intermediate results from the calculation of the distance between two points in the Cartesian coordinate system are displayed in the "Program status".

```
VAR
    Distance,
    x_koord0, x_koord1,
    y_koord0, y_koord1,
    z_koord0, z_koord1 : LREAL;
END_VAR
// ...
Distance := SQRT (_trcVal (_trcVal (x_koord1 - x_koord0)**2) +
                  _trcVal (_trcVal (y_koord1 - y_koord0)**2) +
                  _trcVal (_trcVal (z_koord1 - z_koord0)**2));
```

The function return value is displayed six times in the form "`_TRCVAL = ...`" in the "program status". The sequence of the display corresponds to the sequence of occurrence in the _trcVal function in the formula.

9.22 Creating general references

9.22.1 REF function

Description

The function returns the reference for a variable (address information). This can, for example, be assigned to a variable declared as reference to the data type of the passed actual parameter (REF_TO).

Note

Note for instances of function blocks passed as in/out parameter (VAR_IN_OUT):

- The REF function cannot be used for static variables (VAR) of function blocks imported from technology packages or device-independent libraries.

The restriction does not apply to class instances.

To use this function, you must activate the "Permit object-oriented programming" compiler option, see SIMOTION ST Programming and Operating Manual.

Declaration

```
REF (
    in  : ANY
    )   : REF_TO ANY
```

Input parameter

in

Data type: ANY

The variable for which the reference should be formed

The following are permitted:

- Retentive and non-retentive unit variables or instances of classes or function blocks or their elements declared as such.
- In methods within classes or function blocks:
  - Static variables (VAR) of the higher-level class or function block (including instance variables for classes or function blocks).
  - The THIS keyword. It permits access to the instance variable of the higher-level class or function block.
- In function blocks:
  - Static variables (VAR) of the function block (including instance variables of classes or function blocks).
  - The THIS keyword. Permits access to the instance variable of the function block.

The variable must be able to be read and written.

Not permitted are:

- Variables of technology object data types
- Variables of object-oriented interfaces
- Variables of general references
- Retentive local variables within classes or function blocks
- I/O variables
- Global device variables
- Constants

Return value

Data type: REF_TO ANY

Reference to the variable.

The data type is "Reference to the data type of the input parameter *in*".

9.23 Additional available system functions

In SIMOTION additional system functions are available that are introduced, for example, by the SIMOTION devices and technology objects. The following table lists provides an overview of where these functions are described.

Table 9-17 Overview of additional system functions and system function blocks in SIMOTION ST

System function	Description
System functions of technology objects	SIMOTION Cam Technology Package, System Functions List Manual (reference list) SIMOTION TControl Technology Package List Manual (reference list) Additionally see the function manuals on the technology objects
System functions of SIMOTION devices	List Manual of the SIMOTION devices (reference list)
System functions for controlling axes in accordance with the PLCopen standard	SIMOTION Cam Technology Package, System Functions List Manual (reference list)
Standard functions for controlling I/O modules and drive components	List Manual for the corresponding I/O module and drive component (reference list)

9.24 Application of certain system functions

9.24.1 Programming messages

9.24.1.1 General

You can use the following functions to program messages, e.g. error messages, or check their status:

- _alarmSId (generation of a message without acknowledgment)
- _alarmSqId (generation of a message with acknowledgment)
- _alarmScId (query about message status)

However the prerequisite is an application-specific configured message name.

Note

You can use the *_writeAndSendMessage* function to make user-defined entries in the diagnostic buffer. For a description of this function, refer to the List Manuals for SIMOTION devices.

See also General information for the message programming (Page 356).

9.24.1.2 Overview of the functions

The functions described here can be used in libraries.

During each call, **_alarmSId** generates a **message that does not require acknowledgment**. The message is triggered according to a signal and an associated value can be appended to it. The message is transferred to all display devices registered for this purpose.

During each call, **_alarmSqId** generates a **message that requires acknowledgment**. The message is triggered according to a signal and an associated value can be appended to it. The message is transferred to all display devices registered for this purpose and can be acknowledged at these devices.

Note

No display device registered

If no display device is registered, the acknowledgment status of an Alarm_SQ is not reset when there is a new "incoming" message from the same alarm instance.

In this status, an Alarm_SQ is no longer displayed by the _getPendingAlarms() function.

Therefore, use _alarmScId() to read the status of the message.

You use the following as input signals for the functions:

1. The signal that triggered the message:
   This is interpreted as follows:
   - If the signal represents a rising edge – relative to the last call with this message name – an incoming message is generated. An incoming message is also generated if the signal state is TRUE on the first call with this message name.
   - If the signal represents a falling edge – relative to the last call with this message name – an outgoing message is generated.
2. The message to be compiled:
   It is specified via a unique, project-wide AlarmId (Page 356).
3. Optional associated values provided they have been specified in the message configuration.
   - Independent of the SIMOTON kernel version:
     One associated value of the general data type ANY_NUM or ANY_BIT possible
   - As of version V4.5 of the SIMOTION kernel:
     A maximum of 12 associated values of BYTE (or SINT, USINT) data type is possible; they are passed in an ARRAY [0..11] OF BYTE.
     Associated values with data types that comprise several bytes (e.g. WORD, DINT, REAL) are possible; they reduce the number of possible associated values. The user must ensure that their values are stored in the permissible array elements in accordance with the Big Endian byte order. The AnyType_to_BigByteArray function can be used for this purpose.

The **_alarmScId** function queries the status of a message and its acknowledgment status. The relevant message is specified via a unique AlarmId (Page 356).

For information about the formal structure of functions, see:

- _alarmSId function (Page 357).
- _alarmSqId function (Page 359).
- _alarmScId function (Page 362).

For message names configured application-specifically, see online help.

Note

You should only generate an outgoing message after an incoming one, otherwise an error message is output.

The associated value (optional) must be a variable of an elementary data type. Avoid the use of constant values in the function call!

9.24.1.3 Buffer management of AlarmS

Description

A message list with 40 buffer areas is available for the AlarmS messages. Incoming messages are entered with their ID in this message list. For each outgoing message, an incoming message with the same ID must exist in the message list.

For each of the total 40 list entries there is also a send buffer. This send buffer is used to organize the notification of the registered client (HMI or SIMOTION SCOUT).

Message list and send buffer

Message list and send buffer are used as follows:

Return value		Meaning
Return values for incoming message (function call with rising edge *sig*)		
	16#8002	All entries in the message list are already occupied. Message entry is not made in the message list.
	16#8003	There is still an outgoing message with this ID in the message list and the send buffer is still occupied. Message entry is not made in the message list.
	16#8004	There is already an incoming message for this ID in the message list. Message entry is not made in the message list.
	16#0000	A message with this ID has not been found in the message list and the send buffer is no longer occupied. • ID of the message is entered in the message list. • The associated send buffer is occupied. • The system function returns with return value 16#0000. • The registered clients will be notified. • The send buffer is released again after the successful notification of the clients. • The ID of the message remains as incoming message in the message list.
Return values for outgoing message (function call with falling edge *sig*)		

Return value		Meaning
	16#8003	There is still an incoming message for this ID in the message list and the send buffer is still occupied. Message entry is not made in the message list.
	16#8004	There is already an outgoing message for this ID in the message list. Message entry is not made in the message list.
	16#8007	An incoming entry has not been found for the ID. Message entry is not made in the message list.
	16#0000	An incoming message for the ID has been found in the message list and the send buffer is no longer occupied. • The entry in the message list will be overwritten with the outgoing message. • The associated send buffer is occupied. • The system function returns with return value 16#0000. • The registered clients will be notified. • The send buffer is released again after the successful notification of the clients. • The entry in the message list will be deleted.

9.24.1.4 Example of message generation

The example in the table checks the temperature and generates an incoming message with one associated value which does not have to be acknowledged (e.g. *Temperature too high, incoming*), if the temperature is too high. If the temperature drops below the maximum value specified, an outgoing message is generated (the incoming message disappears).

The message named *SCOUT_alarm_name* has been configured in SIMOTION SCOUT, as for example: `Temperature too high: @1I%2d@ degrees.` A status variable prevents the same message from being repeated. The *handleAlarm* program is assigned to the BackgroundTask.

This example is independent of the SIMOTON kernel version.

Example of message generation

```
INTERFACE
  PROGRAM handleAlarm;
END_INTERFACE

IMPLEMENTATION
  PROGRAM handleAlarm
    VAR
      retVal         : DWORD;         // Return value
      temperature    : INT;           // Status to be queried
      maxTemperature : INT  := 60;    // Signal comparison value for status
      mySignal       : BOOL := FALSE; // Signal yes/no
    END_VAR
    //...
```

Example of message generation

```
    IF temperature > maxTemperature THEN
      IF mySignal = FALSE THEN
        // Incoming message, does not require acknowledgment
        retVal := _alarmSId (
            Sig   := TRUE,
            Ev_id := _alarm.SCOUT_alarm_name,
            sd    := temperature);
        mySignal := TRUE;
      END_IF;
    ELSE
      IF mySignal = TRUE THEN
        // Outgoing message, does not require acknowledgment
        retVal := _alarmSId (
            Sig   := FALSE,
            Ev_id := _alarm.SCOUT_alarm_name,
            sd    := temperature);
        mySignal := FALSE;
      END_IF;
    END_IF;
    //...
  END_PROGRAM
END_IMPLEMENTATION
```

9.24.1.5 Example of messages with several associated values (as of kernel V4.5)

As of version V4.5 of the SIMOTION kernel, several associated values can be specified for message configuring.

The _alarmSId (Page 357) and _alarmSqId (Page 359) functions are passed the associated value in an ARRAY FIELD [0..11] OF BYTE.

Associated values in the ARRAY [0..11] OF BYTE

The following table shows which array elements in the ARRAY [0..11] OF BYTE are assigned the associated value of a specific data type, as well as the addressing of the associated values in the message configuring.

The following applies to an associated value with a data type comprising several bytes:

- It must be stored in consecutive array elements. The first index must be divisible by the number of bytes assigned to the data type.
- It must be stored in the array with big endian byte order (most significant byte at low memory address). The AnyType_to_BigByteArray (Page 396) function can be used for this purpose.

Table 9-18 Ordering of the associated values in the ARRAY [0..11] OF BYTE and access to the message configuring

Data type	Access to the associated values depending on the index of the byte array											
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
BOOL	@1B...	@2B...	@3B...	@4B...	@5B...	@6B...	@7B...	@8B...	@9B...	@10B...	@11B...	@12B...
BYTE, USINT	@1Y...	@2Y...	@3Y...	@4Y...	@5Y...	@6Y...	@7Y...	@8Y...	@9Y...	@10Y...	@11Y...	@12Y...
WORD, UINT	@1W...		@2W...		@3W...		@4W...		@5W...		@6W...	

Data type	Access to the associated values depending on the index of the byte array											
	[0]	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**
INT	@1I...		@2I...		@3I...		@4I...		@5I...		@6I...	
DWORD, UDINT	@1X...				@2X...				@3X...			
DINT	@1D...				@2D...				@3D...			
REAL	@1R...				@2R...				@3R...			
LREAL	@1O...											
Combination examples												
Example 1	@1W...		@3Y...	@4Y...	@2R...							
Example 2	@1X...				@3I...		@7Y...	@8Y...	@3R...			
Example 3	@1W...		@3Y...	@4Y...	@5Y...				@3R...			

For example, an associated value of the INT data type can be stored with indexes [4] and [5] in the array elements. During the message configuring, access is made to this associated value with `@3I...`, i.e. with the format code `%4d` (decimal number with at least 4 digits), the output of this associated value is configured with `@3I%4d@`.

Example

In the message configuring, a *SCOUT_alarm* message is configured with five associated values:

- One associated value of DWORD data type
- One associated value of INT data type
- Two associated values of BYTE data type
- One associated value of REAL data type

The associated values are placed in ARRAY [0..11] OF BYTE passed to the _alarmSId (Page 357) and _alarmSqId (Page 359) functions in accordance with combination example 2 in the above table. The message text can, for example, be configured as follows:
`Value 1 @1X%6X@, value 2 @3I%5d@, value 3 @7Y%2X@, value 4 @8Y%2X@, value 5 @3R%8.2f@`

The following program example shows how the associated values are written in big endian byte order in an ARRAY [0..11] OF BYTE and passed to the _alarmS function. For information about big endian byte ordering, see Converting between any data types and byte arrays (marshalling) (Page 501).

Table 9-19 Example of message programming with several associated values

```
PROGRAM alarm_example
  VAR
    alarm_array : ARRAY [0..11] OF BYTE;
    value_1     : DWORD := 100;
    value_2     : INT   := -10;
    value_3     : BYTE  := 1;
    value_4     : BYTE  := 2;
    value_5     : REAL  := 3.14;
    ret_val     : DWORD;
  END_VAR
```

```
  alarm_array := AnyType_to_BigByteArray (anydata := value_1, offset := 0);
  alarm_array := AnyType_to_BigByteArray (anydata := value_2, offset := 4);
  alarm_array := AnyType_to_BigByteArray (anydata := value_3, offset := 6);
  alarm_array := AnyType_to_BigByteArray (anydata := value_4, offset := 7);
  alarm_array := AnyType_to_BigByteArray (anydata := value_5, offset := 8);
  // Incoming message
  ret_val := _alarmSId (Sig   := TRUE,
                        Ev_id := _alarm.SCOUT_alarm,
                        sd    := alarm_array);
END_PROGRAM
```

9.24.1.6 Checking the error number and status of a message (filtering return values)

_alarmSId and _alarmSqId functions

The return value of the _alarmSId (Page 357) and _alarmSqId (Page 359) functions contains the error number and thus indicates whether an error occurred during execution. As with most system functions, return value = 0 indicates error-free execution.

_alarmScId function

However, the return value of the _alarmScId (Page 362) function indicates both the error number and the status of a message. Therefore, proceed as follows when checking the status with these functions:

1. First, filter the return value with the constant ALARMS_ERROR (= 16#8000).
   This enables you to determine whether an error occurred while the function was being executed. The filter and the error numbers are selected such that they are true when combined in an AND operation.
2. If no error has occurred, you can evaluate the status of the message.

You can find a complete listing of error numbers and message states in the description of the _alarmScId function (Page 362).

Consequently, you can check for errors when the *_alarmScId function* is called (the retVal (Page 362) variable of data type DWORD contains the return value of the function) as follows:

Example of error and status check with the _alarmScId function

```
retVal := _alarmScId (Ev_id := _alarm.SCOUT_alarm_name);
// Error check here
IF (retVal AND ALARMS_ERROR) <> 0 THEN
    // Condition satisfied, therefore an error has occurred.
    // Error number check
    IF retVal = (ALARMS_ERROR OR
        DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID) THEN
        ; // Message number not permitted.
    END_IF;
ELSE
    // Condition not satisfied, therefore no error has occurred.
    // Check of the message and acknowledgment state
    IF retVal = 16#0000 THEN
        ; // Outgoing message, not acknowledged
    ELSIF retVal = ALARMS_STATE THEN
        ; // Incoming message, not acknowledged
    ELSIF retVal = 16#0010 THEN
        ; // Message not available
    ELSIF retVal = (ALARMS_QSTATE OR ALARMS_STATE) THEN
       ; // Incoming message, acknowledged
    END_IF;
    //...
END_IF;
```

Note

You can use constant values and symbolic constants on an equal footing to check for errors, see _alarmScId function (Page 362).

9.24.2 Consistent reading and writing of variables (semaphores)

9.24.2.1 Consistent data access

All accesses to variables of elementary data types (see Section *Elementary data types*) are managed consistently by the system. The system ensures that these variables do not change while you are processing them.

When accessing **global** variables of derived data types (see Section *User-defined data types (UDT)*), the user is responsible for ensuring data consistency when multiple tasks access the same variables (symbolic I/O variables, system variables of SIMOTION devices, system variables of technology objects, global device variables and unit variables, see Section *Variable model*).

Access to **local** variables of derived data types are always consistent, since they can only be used inside the program (or function or function block) in which they are defined.

Note

Consistent data access is always ensured within a task.

9.24.2.2 Semaphores

To ensure consistent reading and writing of global variables, you work with semaphores.

A global variable (e.g. *semaA*) of data type DINT is used as a semaphore. If the variable is an element of an array, the index must be specified when compiling (e.g. a[2]).

Use the following functions to change and test the status of the semaphore:

- _testAndSetSemaphore (sema : DINT) : BOOL
  This function is used to check whether the semaphore is set:
  - Return value TRUE: The semaphore was free and is now set.
  - Return value FALSE: The semaphore is already set.

  After the function ends, the semaphore is always set, either through calling of the function or it was already set. Additional calls of this function (even from other programs) return a value of FALSE, until the _releaseSemaphore (semaA) function is called.
- _releaseSemaphore (sema: DINT) : VOID

For information about the formal structure of functions, see the following sections _testAndSetSemaphore function (Page 418), _releaseSemaphore function (Page 419).

The semaphore is released.

Consistent data access to global variables is ensured under the following conditions:

1. All tasks signal the beginning of an access operation to global variables by setting a semaphore.
2. All tasks signal the end of an access operation to global variables by releasing the semaphore.

9.24.2.3 Example: Consistent data access with semaphores

The example in the table illustrates the use of semaphores in a program that reads data and a program that writes data.

Table 9-20 Example for ensuring consistent access to global variables using semaphores

```
IMPLEMENTATION
    VAR_GLOBAL
        myArray : ARRAY [0..1] OF DINT;
        semaA : DINT;
    END_VAR

    PROGRAM Writer
        // Consistent writing of variables
        IF _testAndSetSemaphore(sema := semaA) THEN
            myArray[0] := 18;
            myArray[1] := 19;
            _releaseSemaphore(sema := semaA);
            // The semaphore must be released in the TRUE
            // branch of the query; this ensures that
            // it is only released when it has been
            // reset.
        ELSE
            ; // Error handling
        END_IF;
        // _releaseSemaphore(sema := semaA);
        // The release would be incorrect at this point;
        // the semaphore would always be released.
    END_PROGRAM
    PROGRAM Reader
        VAR
            var0        : DINT;
            var1        : DINT;
        END_VAR

        // Consistent reading of variables
        IF _testAndSetSemaphore(sema := semaA) THEN
            var0 := myArray[0];
            var1 := myArray[1];
            _releaseSemaphore(sema := semaA);
            // The semaphore must be released in the TRUE
            // branch of the query; this ensures that
            // it is only released when it has been
            // reset.
        ELSE
            ; // Error handling
        END_IF;
        // _releaseSemaphore(sema := semaA);
        // The release would be incorrect at this point;
        // the semaphore would always be released.
    END_PROGRAM

END_IMPLEMENTATION
```

9.24.3 Data backup and initialization from user program

9.24.3.1 Data backup and data initialization from user program - functions and instructions

From a user program, it is possible to save, load, or initialize the values of the following variables in data sets:

- Non-retentive or retentive unit variables of the interface or implementation section of a unit (e.g. ST source file, MCC unit).
- Non-retentive or retentive global device variables.

Various functions are available for this:

- _saveUnitDataSet (Page 428): The values are stored as a binary data set.
- _loadUnitDataSet (Page 431): The values are loaded from a binary data set saved with *_saveUnitDataSet*.
  Loading the data set is only possible if since saving the data set the version IDs of all data segments to be loaded have remained unchanged.
  Please also pay attention to the note below.
- _exportUnitDataSet (Page 434): The values are exported in XML format as ZIP archive (file name *.dat).
- _importUnitDataSet (Page 437): The values are imported from a data set that was exported in XML format as ZIP archive (file name *.dat) with *_exportUnitDataSet*.
  Importing the data set is also possible if, since the data set was saved, the version ID of a data segment to be loaded has changed:
  - Variables that no longer exist are ignored.
  - The value of added variables remains unchanged.
    You can, for example, initialize the relevant data segments with *_resetUnitData* before importing a data set to avoid unwanted values in variables.
  - In the case of a variable with a changed data type, the value is transferred if it can be converted to the new data type; otherwise the value of the variable is retained.
- _deleteUnitDataSet (Page 440): A single data set is deleted.
- _checkExistingUnitDataSet (Page 443): A check determines whether the specified data set exists on the storage medium.
- _deleteAllUnitDataSets (Page 445): All data sets are deleted.
- _resetUnitData: The values of the variables are initialized, see *System Functions of Devices* List Manual.

For data segments and their version ID, see Section "Version ID of global variables and their initialization during download" in the Programming Manuals, e.g. SIMOTION ST.

The description for *_resetUnitData* can be found in the List Manual (reference list) for the system functions of the SIMOTION devices.

The input parameters (Page 482) and the return value (Page 483) of the functions are explained in detail in the following sections.

NOTICE
Data loss possible If the data structure of a data segment is changed in a unit or in the global device variables, the following applies to loading the project into the SIMOTION device: • All temporarily backed-up data sets are deleted. • None of the binary data sets that have been permanently saved (with *_saveUnitDataSet*) can be read for this data segment any longer.

Up to 16 data backups can run simultaneously on a device.

The number of data sets that can be saved - up to 1_000_000 - depends on the available memory space.

Pay attention to consistency of the data to be backed up or exported, see Consistent reading and writing of variables (semaphores) (Page 478).

Note

You can upload data sets saved with *_saveUnitDataSet* or *_exportUnitDataSet* from the SIMOTION device. Use the **Save variables** function in SIMOTION SCOUT. The data sets saved with *_saveUnitDataSet* are automatically converted to XML format.

You can use the **Restore variables** function to download these data sets and variables back to the SIMOTION device. For this purpose select the relevant SIMOTION device in the project navigator and select the function from the context menu. For more information, refer to the SIMOTION SCOUT Configuration Manual.

This makes it possible, for example, to obtain the data saved with *_saveUnitDataSet*, even if it is initialized by a project download or if it becomes unusable (e.g. due to a SIMOTION version change).

9.24.3.2 Input parameters

The most important parameters are briefly outlined below. For a more detailed description of the individual functions and their parameters, refer to Backing up data from the user program (Page 427).

- *unitName*: Name of the unit (e.g. ST source file, MCC unit)
  If '*_device*' is specified, the data backup function is applied to global device variables (possible with _saveUnitDataSet).
- *id*: Data set number
  Specifying a data set number as an index enables several versions of the unit variables to be saved and then downloaded selectively.
  id < 1_000_000

- *storageType*: Specifying the storage location allows you to select the following:
  - TEMPORARY_STORAGE: Data is saved to the RAM disk (deleted after a power failure).
  - PERMANENT_STORAGE: Data is saved to a memory card (and retained after a power failure).

  Note also the additional information on the Storage location and memory requirement (Page 484).
- *nextCommand*: Step enabling condition
  Specifying the step enabling conditions enables you to select from the following options:
  - Execute next command in the ST source file immediately (IMMEDIATELY).
  - Wait until command is done (WHEN_COMMAND_DONE).

  Note also the additional information on the Step enabling condition (Page 485).
- *dataScope*
  It enables the section of the unit to which the data backup function is applied to be selected.
  - _INTERFACE: Function is applied to the interface section of a unit.
  - _IMPLEMENTATION: Function is applied to the implementation section of a unit.
  - _INTERFACE_AND_IMPLEMENTATION: Function is applied to the interface and implementation section of a unit.

  If *unitName* = '*_device*' is specified, only the values _INTERFACE or _INTERFACE_AND_IMPLEMENTATION are permissible for *dataScope*.
- *kindOfData*:
  It permits selection of whether the data backup function will be applied to retentive or retentive global variables.
  - NO_RETAIN_GLOBAL: Function is applied to non-retentive global variables.
  - _RETAIN: Function is applied to retentive global variables.
  - ALL_GLOBAL: Function is applied to retentive and non-retentive global variables.

  If retentive and non-retentive variables are stored in a data set, it is possible to load or import retentive or non-retentive variables selectively.

9.24.3.3 Return value

The return value is a structure of data type *StructRetUnitDataSetCommand*. It comprises the following:

- A *functionResult* component: *EnumDeviceUnitDataSetCommand*
  This supplies information on errors and the current status.
- A *handle* component: UDINT
  This allows you to check the current status of a data backup function by means of the *_getStateOfUnitDataSetCommand* function (this is required for step enabling condition IMMEDIATELY).

9.24.3.4 Storage location and memory requirement

You define the storage location with input parameter *storageType*:

- TEMPORARY_STORAGE: Data sets are saved to the RAM disk.
- PERMANENT_STORAGE: Data sets are saved to the Memory Card (under \USER\SIMOTION\USER_DIR\).

The number of data sets that can be saved depends on the available memory space at the respective memory location. Information on the available memory space can be obtained by means of the Device diagnostics, **System utilization** tab (see online help).

Note

Do not fill the entire available memory space with data sets! Otherwise, it will no longer be possible to download a project to the target system or copy RAM to ROM if project data is increased!

You can estimate the required memory space for binary data sets (saved with _saveUnitDataSet (Page 428)) with the following information:

- Elementary data types occupy their natural data width on the memory (see Table *Bit widths and value ranges of the elementary data types* in Chapter *Elementary data types*; data type BOOL occupies 1 byte).
- Additional memory space is required due to:
  - Adaptation of the memory addresses to word or double-word boundaries
  - Consistency information (approx. 100 bytes per data set)
  - Usual supplementary data of a file system (e.g. sector headers, directory, occupy whole sectors only)

For data sets exported in XML format (with _exportUnitDataSet (Page 434)), the memory requirement is much higher and cannot be ascertained in this way.

9.24.3.5 Step enabling condition

General information

The step enabling condition is indicated in input parameter *nextCommand*.

- **Synchronous call**
  The function is called with parameter *nextCommand* := WHEN_COMMAND_DONE. The next command in the program source when the function has been completed.
  The return value contains the result of the executed function in its *functionResult* component (see example).
  This procedure is mainly used in sequential tasks (e.g. MotionTasks).
- **Asynchronous call**
  The execution of this function may take quite a long time. Therefore, the time watchdog might respond in connection with cyclic tasks (e.g. BackgroundTask).
  For this reason, the function can also be executed asynchronously by setting parameter *nextCommand* := IMMEDIATELY. In this case, the function is started, and then the next command in the source is immediately processed.
  From the return value you can see:
  - If the start was successful (component *functionResult* = DONE)
  - A handle for further status query (*handle* component)

  If the command start was successful, you must check the current status of the data backup function using the _getStateOfUnitDataSetCommand (Page 442) function and the handle until the result is something other than ACTIVE (see example).

Example of a synchronous call (in sequential tasks)

The data backup function _loadUnitDataSet (Page 431) is called with the step enabling condition WHEN_COMMAND_DONE.

```
VAR_GLOBAL
    ds_ret : StructRetUnitDataSetCommand;
    error  : BOOL := FALSE;
END_VAR

PROGRAM save_data_seq
    // Program is assigned to a sequential task.
    // Function is executed synchronously:
    ds_ret := _loadUnitDataSet (
            unitName := 'ds3',
            id := 1,
            storageType := TEMPORARY_STORAGE,
            nextCommand := WHEN_COMMAND_DONE);
    // Function completed, evaluate result
    IF (ds_ret.functionResult <> DONE) THEN
        error := TRUE;    // Fehler
    END_IF;
END_PROGRAM
```

Example of an asynchronous call (in cyclic tasks)

The data backup function _saveUnitDataSet (Page 428) is called with the step enabling condition IMMEDIATELY. The _getStateOfUnitDataSetCommand (Page 442) is then called until the _saveUnitDataSet function has been completed

```
VAR_GLOBAL
    error : BOOL := FALSE;
    ds_rslt      : EnumDeviceUnitDataSetCommand;
    ds_ret       : StructRetUnitDataSetCommand;
    cmd_busy     : BOOL := FALSE;
    cmd_done     : BOOL := FALSE;
END_VAR

PROGRAM save_data_cycl
    // Program is assigned to a cyclic task.
    IF NOT cmd_busy THEN
        cmd_busy := TRUE;
        // Function is executed asynchronously:
        ds_ret := _saveUnitDataSet (
                unitName := 'ds1',
                id := 1,
                storageType := TEMPORARY_STORAGE,
                overwrite   := TRUE,
                nextCommand := IMMEDIATELY);
        IF (ds_ret.functionResult <> DONE) THEN
            cmd_busy := FALSE;
            error := TRUE; // Start of the function has failed
            // (e.g. too many services)
        END_IF;
    ELSE
        // Function is running, wait for result:
        ds_rslt := _getStateOfUnitDataSetCommand (
                handle := ds_ret.handle);
        IF (ds_rslt <> ACTIVE) THEN
            cmd_busy := FALSE;
            IF (ds_rslt = DONE) THEN
                cmd_done := TRUE;
                // Function terminated successfully
            ELSE
                error := TRUE;
                // Function has failed
            END_IF;
        END_IF;
    END_IF;
END_PROGRAM
```

9.24.4 Accessing files from a user program (as of V4.4)

As of version 4.4 of the SIMOTION Kernel, the user can access the file system of the SIMOTION device via a program. The user can access the permanent storage medium (e.g. the memory card) or the temporary storage medium (e.g. the RAM disk) and read or write files. The user can also manage the files (e.g. copy, rename and delete).

The user can only use selected areas of the file system on the applicable storage medium so that SIMOTION function is not impaired. These areas are addressed by symbolic names.

The following sections describe how to:

- read or write files (Page 487), and
- manage files (Page 497).

Note

Memory cards are not designed to be rewritten an unlimited number of times. With this in mind, you should avoid writing user data from the application to the memory card cyclically. Depending on the system, a write operation from the application may trigger one or more write operations on the memory card. Therefore, we recommend you adopt a conservative approach in terms of the number of writing processes. In other words, do not perform more than 100,000 write access instances from the user program over the estimated service life of the application.

Never switch off a SIMOTION device during write accesses to the memory card. If the SIMOTION device is switched off during write accesses, this can result in destruction of the data and, in the worst case, damage the file system (FAT Table = table of contents) on the memory card.

Follow the instructions in the section "Recommended method of handling CF cards" in Commissioning and Hardware Installation Manual SIMOTION D4x5-2.

9.24.4.1 Reading and writing files from a user program (as of V4.4)

As of version 4.4 of the SIMOTION Kernel, it is possible to read and write files from a user program. These files can be on the permanent storage medium (e.g. memory card) or the temporary storage medium (e.g. RAM disk). The following function blocks are provided for this purpose. Their functionality and use is described below.

Brief description of function blocks

- _filehandle function block
  See the information about "Filehandle" in this section.
- _fileOpen function block
  This function block opens the file specified in the parameters STORAGETYPE, LOCATION and FILENAME and links them to the instance of the _filehandle function block transferred in the HANDLE parameter.
  For information about the STORAGETYPE, LOCATION and FILENAME parameters, see "General information about the storage locations of a file" in the following section (Page 497).
  Specify using the OPENFILE parameter whether the file should only be opened with write authorization or with read and write authorization. A file can either be opened multiple times for read-only access or only once exclusively for read and write access.
  If a file that is not present is opened for read and write, it is created with a fixed length (as well as the path, if not present). The fixed length is specified in the FILESIZE parameter. If the file already exists under the specified path, this parameter is ignored.
  A maximum of five files can be opened at the same time.
- _fileSetPosition function block
  This function block determines the position within an open text file.
  This file is identified via the _filehandle function block instance transferred in the HANDLE parameter.
  The position is specified relative to the FILEPOSITION parameter (FILEPOSITIONOFFSET parameter).
- _fileRead function block
  This function block reads binary data from the current position of the open file and copies it to an ARRAY OF BYTE (BUFFER parameter).
  The file to be read is identified via the _filehandle function block instance transferred in the HANDLE parameter.
  The number of bytes to be read is specified in the DATALENGTH parameter. The BUFFERPOSITION parameter specifies the index of the ARRAY OF BYTE from which the data is copied into the array.
- _fileWrite function block
  This function block copies binary data from an ARRAY OF BYTE (BUFFER parameter) and writes it into the open file from the current position.
  The file to be written is identified via the _filehandle function block instance transferred in the HANDLE parameter.
  The number of bytes to be written is specified in the DATALENGTH parameter. The BUFFERPOSITION parameter specifies the index of the ARRAY OF BYTE from which the data is copied from the array.
- _fileReadLn function block
  This function block reads data from the current position up to the end of line (CR/LF - $0D$0A) from the open text file and copies it into a string (LINE parameter).
  The text file to be read is identified via the _filehandle function block instance transferred in the HANDLE parameter.

- _fileWriteLn function block
  This function block writes a string (LINE parameter) into the open text file from the current position and ends the line with CR/LF ($0D$0A).
  The text file to be written is identified via the _filehandle function block instance transferred in the HANDLE parameter.
  The written data is visible in the WRITTENLINE parameter.
- _fileClose function block
  This function closes the open file and releases the system resources.
  The file to be closed is identified via the _filehandle function block instance transferred in the HANDLE parameter.
  If the file was open for read and write access, potential changes are saved on the storage medium.

The syntax of these function blocks is described in detail in the "System Functions/Variables Devices" List Manual (reference list) and in the online help (see index).

Operation of the function blocks

The interfaces of the function blocks behave in accordance with the PLCopen specification "Technical Specification, Function blocks for motion control, Version 1.1".

The rules described in the "SIMOTION PLCopen Blocks" Function Manual in Chapter "General rules for the PLCopen FB interface" apply for the PLCopen function blocks.

The function blocks work asynchronously, i.e. after calling an instance of these function blocks, they are processed in the background.

1. They declare an instance of the function block, and class this instance in a program which is integrated into a cyclic task (e.g. BackgroundTask).
   As long as the value is set to FALSE in the EXECUTE input parameter, the instance of the function block is not executed.
2. A positive edge in the EXECUTE input parameter starts the function block. The BUSY output parameter is set to TRUE.
   As long as the job is being processed, then BUSY = TRUE. Further positive edges in the EXECUTE input parameter are then ignored and an instance of the function block can no longer be called.
   The instance of the function block is processed in the background.
3. When the job is finished, BUSY is set to FALSE.
   - If execution was successful, the DONE output parameter is set to TRUE.
   - If execution completed with errors, the ERROR output parameter is set to TRUE and the cause of error is displayed in the ERRORID output parameter.

   Only one of the three output parameters BUSY, DONE or ERROR can have the value TRUE at any one time.
4. A negative edge in the EXECUTE input parameter resets the DONE and ERROR output parameters to FALSE.
   If the negative edge occurs at EXECUTE while a job is active (BUSY = TRUE), the running job is not aborted. After the job has finished, the DONE or ERROR output parameters are, however, only in TRUE status for one cycle time.

The instances of the function blocks can also be called in sequential tasks (e.g. MotionTasks). However, the user must ensure the cyclic execution. Refer to the information in "Basic procedure for reading and writing files" later in this section.

Filehandle

Filehandle is an instance of the _filehandle function block. It is transferred to the function blocks for open, read, write and close, and identifies the file to be read or written uniquely. It provides the system resources to access the file system and its structure.

Calling an instance of the _fileOpen function block links the filehandle to a file in the file system The system resources are assigned.

The location at which the instance of the _filehandle function block has been declared determines its initialization behavior and therefore the life of the system resources. See also the section "Initialization of instances of function blocks (FBs)" in the SIMOTION ST Programming Manual.

The system resources are released again by closing the file with an instance of the _fileClose function block.

Note

- Accesses to the storage medium are buffered. For this reason, e.g. for write accesses, only once the file has been closed successfully with an instance of the _fileClose function block can it be ensured that the data have been written in full to the storage medium.
- If the instance of the _filehandle function block is declared as a static variable that is saved in the user memory of the task, the following applies:
  On transition to STOP mode, an existing job is processed first before the system resources are released. On rapid transition to RUN mode, it may therefore be the case that the system resources are still assigned.
  For information about variables in the user memory of the task, see section "Memory areas of the variable types" in the SIMOTION ST Programming Manual.

A separate call of an instance of the _filehandle function block provides information about the associated filehandle and the storage medium (available memory and number of open files).

The syntax of the _filehandle function block is described in detail in the "System Functions/ Variables Devices" List Manual (reference list) and in the online help (see index).

Basic procedure for reading and writing files

For an asynchronous call in a cyclic task (e.g. BackgroundTask)

The following basic procedure is recommended:

1. Declare instances of the _filehandle, _fileOpen and _fileClose function blocks and the other necessary function blocks.
2. Provide a start signal. For this purpose declare, e.g. a unit variable of data type BOOL with initialization value FALSE.

3. Call the instance of the _fileOpen function block first. Assign these values to the following parameters:
   - HANDLE: the instance of the _filehandle function block.
   - EXECUTE: the start signal.
   - STORAGETPYE, LOCATION and FILENAME: as per the file in the file system.
     See "General information about the storage locations of a file" in the next section (Page 497).
   - OPENFILE: READONLY or READWRITE.
4. Call the required instances of the function blocks to read write or position. Assign these values to the following parameters:
   - HANDLE: the instance of the _filehandle function block.
   - EXECUTE: the DONE output parameter of the preceding instance.
5. Release the system resources. To do so, call the instance of the _fileClose function block. Assign these values to the following parameters:
   - HANDLE: the instance of the _filehandle function block.
   - EXECUTE: the DONE output parameter of the preceding instance.
6. Assign the program to a cyclic task in the execution system (e.g. BackgroundTask).
7. In RUN operating mode, assign the value TRUE to the start signal at the required time. Execution starts. The instances of the function block are processed sequentially in the background.

The following examples 1 and 2 explain this procedure.

For a synchronous call in a sequential task (e.g. MotionTask)

In order to use the function blocks in sequential tasks, proceed in a similar way as for asynchronous execution.

However, the user must ensure the cyclic execution, e.g through the following procedure:

1. Include each used instance of a function block in a loop (e.g. REPEAT ... END_REPEAT).
2. Also include a call of the system function _waitTime (Page 450) with the parameter timeValue = T#0s in each of these loops.
   This ensures that the next MotionTask is executed. Also refer to information on "Cyclic MotionTasks" in Section "Time allocation in the round robin execution level" (Page 273).
3. Use a suitable expression with the output parameters BUSY, DONE or ERROR as an abort condition for the loop.

Assign the program to a sequential task in the execution system (e.g. StartupTask, MotionTask).

This procedure is explained in the following example 3.

Examples

Example 1: Create file and write line (asynchronous call)

This example shows how to create a file and write a line in the file.

The program is called in a cyclic task (e.g. BackgroundTask). A positive edge at the bWrite variable starts execution.

```
INTERFACE
    PROGRAM WriteLineToFile;
    VAR_GLOBAL
        bWrite : BOOL := FALSE;
    END_VAR
END_INTERFACE

IMPLEMENTATION
  PROGRAM WriteLineToFile
    VAR
        myFilehandle    : _filehandle;
        myOpenFileFB    : _fileOpen;
        myWriteLnFileFB : _fileWriteLn;
        myCloseFileFB   : _fileClose;
    END_VAR

    myOpenFileFB (
        EXECUTE  := bWrite,
        STORAGETYPE := CARD,
        LOCATION := USERFILES,
        FILENAME := 'MyData/file_name.txt',
        OPENFILE := READWRITE,
        FILESIZE := 512,
        HANDLE   := myFilehandle);

    myWriteLnFileFB (
        EXECUTE := myOpenFileFB.DONE,
        LINE    := 'This is the text that is written.',
        HANDLE  := myFilehandle);

    myCloseFileFB (
        EXECUTE := myWriteLnFileFB.DONE,
        HANDLE  := myFilehandle);
  END_PROGRAM
END_IMPLEMENTATION
```

Example 2: Open file and read line (asynchronous call)

This example shows how to open an available file and read a line from a specific position.

The program is called in a cyclic task (e.g. BackgroundTask). A positive edge at the bRead variable starts execution.

```
INTERFACE
    PROGRAM ReadLineFromFile;
    VAR_GLOBAL
        bRead : BOOL := FALSE;
    END_VAR
END_INTERFACE

IMPLEMENTATION
  PROGRAM ReadLineFromFile
    VAR
        myFilehandle   : _filehandle;
        myOpenFileFB   : _fileOpen;
        myReadLnFileFB : _fileReadLn;
        mySetFilePosFB : _fileSetPosition;
        myCloseFileFB  : _fileClose;
        lineBuffer     : STRING[254];
    END_VAR

    myOpenFileFB (
        EXECUTE  := bRead,
        STORAGETYPE := CARD,
        LOCATION := USERFILES,
        FILENAME := 'MyData/file_name.txt',
        HANDLE   := myFilehandle);

    mySetFilePosFB (
        EXECUTE := myOpenFileFB.DONE,
        FILEPOSITION := BEGINNING_OF_FILE,
        FILEPOSITIONOFFSET := 12,
        HANDLE  := myFilehandle);

    myReadLnFileFB (
        EXECUTE := mySetFilePosFB.DONE,
        HANDLE  := myFilehandle,
        LINE    => lineBuffer);

    myCloseFileFB (
        EXECUTE := myReadLnFileFB.DONE,
        HANDLE  := myFilehandle);
  END_PROGRAM
END_IMPLEMENTATION
```

Example 3: Create file and write lines (synchronous call)

This example shows how to create a file and write some lines in the file.

The program is called in a sequential task (e.g. StartupTask, MotionTask). A positive edge at the bWrite variable starts execution.

```
INTERFACE
    PROGRAM writeFile_From_MotionTask;
    VAR_GLOBAL
        bWrite : BOOL := FALSE;
    END_VAR
END_INTERFACE

IMPLEMENTATION
  FUNCTION_BLOCK write
    VAR_IN_OUT
        execute : BOOL;
    END_VAR

    VAR
        mydret           : DINT;
        myFilehandle     : _filehandle;
        myOpenFileFB     : _fileOpen;
        myWriteLnFileFB  : _fileWriteLn;
        myCloseFileFB    : _fileClose;
        myOpenFileFB2    : _fileOpen;
        myWriteLnFileFB2 : _fileWriteLn;
        myCloseFileFB2   : _fileClose;
        myPosFileFB      : _fileSetPosition;
    END_VAR

    REPEAT
        myOpenFileFB (
            EXECUTE     := execute,
            STORAGETYPE := CARD,
            LOCATION    := USERFILES,
            FILENAME    := 'MyData2/file_name2.txt',
            OPENFILE    := READWRITE,
            FILESIZE    := 512,
            HANDLE      := myFilehandle);
        IF execute = TRUE THEN
            execute := FALSE;
        END_IF;
        mydret := _waittime (timeValue := T#0ms);
        UNTIL myOpenFileFB.DONE OR myOpenFileFB.ERROR
    END_REPEAT;

    REPEAT
        myWriteLnFileFB (
            EXECUTE := myOpenFileFB.DONE,
            LINE    := 'This is the text that is written.',
            HANDLE  := myFilehandle);
        mydret := _waittime (timeValue := T#0ms);
        UNTIL myWriteLnFileFB.DONE OR myWriteLnFileFB.ERROR
    END_REPEAT;

    REPEAT
        myCloseFileFB (
            EXECUTE  := myWriteLnFileFB.DONE,
            HANDLE   := myFilehandle);
        mydret := _waittime (timeValue := T#0ms);
```

```
      UNTIL myCloseFileFB.DONE OR myCloseFileFB.ERROR
  END_REPEAT;

  REPEAT
      myOpenFileFB2 (
          EXECUTE     := myCloseFileFB.DONE,
          STORAGETYPE := CARD,
          LOCATION    := USERFILES,
          FILENAME    := 'MyData2/file_name2.txt',
          OPENFILE    := READWRITE,
          FILESIZE    := 512,
          HANDLE      := myFilehandle);
      mydret := _waittime (timeValue := T#0ms);
      UNTIL myOpenFileFB2.DONE OR myOpenFileFB2.ERROR
  END_REPEAT;

  REPEAT
      myPosFileFB (
          EXECUTE            := myOpenFileFB2.DONE,
          FILEPOSITION       := BEGINNING_OF_FILE,
          FILEPOSITIONOFFSET := 50,
          HANDLE             := myFilehandle);
      mydret := _waittime (timeValue := T#0ms);
      UNTIL myPosFileFB.DONE OR myPosFileFB.ERROR
  END_REPEAT;

  REPEAT
      myWriteLnFileFB2(
          EXECUTE := myPosFileFB.DONE,
          LINE    := 'This is the second text.',
          HANDLE  := myFilehandle);
      mydret := _waittime (timeValue := T#0ms);
      UNTIL myWriteLnFileFB2.DONE OR myWriteLnFileFB2.ERROR
  END_REPEAT;

  REPEAT
      myCloseFileFB2(
          EXECUTE := myWriteLnFileFB2.DONE,
          HANDLE  := myFilehandle);
      mydret := _waittime (timeValue := T#0ms);
      UNTIL myCloseFileFB2.DONE OR myCloseFileFB2.ERROR
  END_REPEAT;
END_FUNCTION_BLOCK

PROGRAM writeFile_From_MotionTask
  VAR
      mywrite : write;
      biwrite : BOOL;
  END_VAR
  bwrite := TRUE;
  IF bwrite THEN
      bwrite  := FALSE;
      biwrite := TRUE;
      mywrite (execute := biwrite);
  END_IF;
END_PROGRAM
```

```
END_IMPLEMENTATION
```

9.24.4.2 Managing files from a user program (as of V4.4)

Files can be managed from a user program (e.g. copy, rename and delete) as of version 4.4 of the SIMOTION Kernel. The files can be on the permanent storage medium (e.g. memory card) or the temporary storage medium (e.g. RAM disk). The following function blocks are provided for this purpose. Their functionality and use is described below.

Brief description of function blocks

- _fileCopy function block
  This function block creates a copy of a file.
  The file to be copied is specified by the SOURCESTORAGETYPE, SOURCELOCATION and SOURCEFILENAME parameters.
  The copy destination is specified by the DESTINATIONSTORAGETYPE, DESTINATIONLOCATION and DESTINATIONFILENAME parameters.
- _fileRename function block
  This function block renames a file inside a directory.
  The file to be renamed is specified by the STORAGETYPE, LOCATION and FILENAME parameters.
  Enter the new file name in the NEWFILENAME parameter without specifying the path.
- _fileDelete function block
  This function block deletes a file from a folder.
  The file to be deleted is specified by the STORAGETYPE, LOCATION and FILENAME parameters.
  The file must not be open. If necessary, the _fileClose function block must be used first.
- _directoryPathDelete function block
  This function block deletes a folder including all sub-folders and their contents.
  The folder to be deleted is specified by the STORAGETYPE, LOCATION and DIRECTORYPATHNAME parameters.
- _getStateOfFile function block
  This function block provides status information about a file or a folder (e.g. size and modified date).
  The file or the folder is specified by the STORAGETYPE, LOCATION and FILENAME parameters.

The syntax of these function blocks is described in detail in the "System Functions/Variables Devices" List Manual (reference list) and in the online help (see index).

General information about the storage locations of a file

For the above function blocks and the _fileOpen (Page 487) function block, a file is specified in the file system via three entries and the associated parameters.

1. The storage medium (e.g. STORAGETYPE parameter).
   The following values are permissible:
   - RAMDISK: temporary, non-retentive storage medium, e.g. RAM disk.
   - CARD: permanent storage medium, e.g. memory card.

   Note

   Files that have been created using the above function blocks or the _fileOpen function block on the temporary storage medium (e.g. RAM disk) are not copied to the permanent storage medium (e.g. memory card) using the "Copy RAM to ROM" SIMOTION SCOUT function.

2. The location on the storage medium (e.g. LOCATION parameter).
   The files are stored in the selected folders on the storage medium. These folders are selected via symbolic names. This ensures that the SIMOTION functions are not affected.
   The following values (symbolic names) are permitted:
   - USERFILES
     Corresponds to path: '\USER\SIMOTION\HMI\USERFILES'
   - USERLOG
     Corresponds to path: '\USER\SIMOTION\HMI\USERLOG'
   - USERDATASETS
     Corresponds to path: '\USER\SIMOTION\USER_DIR\UPP\UNITDS'
     All data and sub-folders in this folder are deleted when the following SIMOTION SCOUT function is used: "Delete user data on card" when the unit data sets checkbox is activated.
   - USERWEB
     Corresponds to path: '\USER\SIMOTION\HMI\FILES'
   - USERTEMP
     Corresponds to path: '\USER\SIMOTION\USER_DIR\USERTEMP'
     All data and sub-folders in this folder are deleted when the following SIMOTION SCOUT function is used: "Delete user data on card"

   Any folder structure can be created beneath the specified folder.

3. The path and the file name (e.g. FILENAME parameter)
   This allows files to be accessed in any folder tree beneath the folders selected using LOCATION.
   Permissible characters for file name and path:
   - All displayable characters from the ASCII character set are permitted as name of the file and sub-folders, except the following symbols:
     '"'($22), '*'($2A), '/'($2F), ':'($3A), '<'($3C), '>'($3E), '?'($3F), '\'($5C), '|'($7C).
   - The following symbols may be used as separator between the folder name and file name:
     '/'($2F), '\'($5C).

Operation of the function blocks

The function blocks are designed for asynchronous calls.

For information about this functionality, see the previous section (Page 487).

Basic procedure to manage files

The following basic procedure is recommended:

1. Declare instances of the required function blocks.
2. Provide a start signal. For this purpose declare, e.g. a unit variable of data type BOOL with initialization value FALSE.
3. Call the required instances of the function blocks sequentially. Assign the following values to the EXECUTE parameter:
   - For the first instance: the start signal.
   - For the subsequent instances: the DONE output parameter of the preceding instance in each case.
4. Assign the program to a cyclic task in the execution system (e.g. BackgroundTask).
5. In RUN operating mode, assign the value TRUE to the start signal at the required time. Execution starts. The instances of the function block are processed sequentially in the background.

Example

This example demonstrates the following file management operations:

1. Rename a file
2. Copy the file from the permanent storage medium (e.g. memory card) to the temporary storage medium (e.g. RAM disk)
3. Delete the original file from the permanent storage medium

The program is called in a cyclic task. A positive edge at the bStart variable starts execution.

```
INTERFACE
    PROGRAM fileSystemOperation;
    VAR_GLOBAL
        bStart : BOOL := FALSE;
    END_VAR
END_INTERFACE

IMPLEMENTATION
  PROGRAM fileSystemOperation
    VAR
        myDeleteFileFB     : _fileDelete;
        myRenameFileFB     : _fileRename;
        myCopyFileFB       : _fileCopy;
        myGetStateOfFileFB : _getStateOfFile;
    END_VAR

    myRenameFileFB (
            EXECUTE  := bStart,
            STORAGETYPE := CARD,
            LOCATION := USERFILES,
            FILENAME := 'MyData/file_name.txt',
            NEWFILENAME := 'new_name.txt');

    myCopyFileFB (
            EXECUTE := myRenameFileFB.DONE,
            SOURCESTORAGETYPE := CARD,
            SOURCELOCATION    := USERFILES,
            SOURCEFILENAME    := 'MyData/file_name.txt',
            DESTINATIONSTORAGETYPE := RAMDISK,
            DESTINATIONLOCATION    := USERFILES,
            DESTINATIONFILENAME    := 'MyTemplates/file.txt');

    myGetStateOfFileFB (
            EXECUTE  := myRenameFileFB.DONE,
            STORAGETYPE := CARD,
            LOCATION := USERFILES,
            FILENAME := 'MyData/new_name.txt');

    IF myGetStateOfFileFB.FILEEXISTING = YES THEN
        myDeleteFileFB (
                EXECUTE  := myGetStateOfFileFB.DONE,
                STORAGETYPE := CARD,
                LOCATION := USERFILES,
                FILENAME := 'MyData/new_name.txt');
    END_IF;
  END_PROGRAM
END_IMPLEMENTATION
```

9.24.5 Converting between any data types and byte arrays (marshalling)

Conversion of variables of any data type to byte arrays, and vice versa, is frequently used to create defined transmission formats for data exchange between different devices (see also section Communication functions (Page 504)).

Variables of any data type (elementary data types, standard data types of technology packages and devices, and user-defined data types) can be converted to byte arrays, and vice versa, using the following functions:

- AnyType_to_BigByteArray (Page 396)
- AnyType_to_LittleByteArray (Page 396)
- BigByteArray_to_AnyType (Page 398)
- LittleByteArray_to_AnyType (Page 398)

For all functions, an offset can be specified optionally for the first element to be assigned or evaluated in the byte array.

A distinction is made as to:

- Conversion direction (to or from byte arrays)
- Byte order in the array (see table):
  - Big endian: Most significant byte at low memory address (Motorola, SUN Sparc, SIMATIC S7)
  - Little endian: Least significant byte at low memory address (Intel, DEC Alpha)

Table 9-21 Examples of byte order (big endian and little endian)

Address	Number 34677374 = 16#2_11_22_7E (data type UDINT)		Character string "Byte" (data type DWORD)	
	Big endian	Little endian	Big endian	Little endian
2#...11	16#7E = 126	16#02 = 2	16#65 = "e"	16#42 = "B"
2#...10	16#22 = 34	16#11 = 17	16#74 = "t"	16#79 = "y"
2#...01	16#11 = 17	16#22 = 34	16#79 = "y"	16#74 = "t"
2#...00	16#02 = 2	16#7E = 126	16#42 = "B"	16#65 = "e"

Note

TO data types cannot be converted. See section “Conversion of technology object data types” (Page 408) for conversion of TO data types.

If structures and arrays are to be converted, consistency is guaranteed only at the elementary variable level. Any higher level of consistency must be ensured by the user (see the following sections:

- "Consistent reading and writing of variables (semaphores)" (Page 478) and
- Consistent access to global variables of derived data types (UDT) (Page 417).

The following data types are not portable convertible, i.e their transmission formats are not defined across systems. Conversions between SIMOTION devices are possible without any restrictions:

- Time types: For transmission format, see Table
- Enumerators (enumeration data types)

When converting these data types, the compiler outputs a warning (16013).

Table 9-22 Transmission formats of time data types for marshalling functions

Data type	Transmission format
TIME	Time specification, in ms (UDINT)
TIME_OF_DAY (TOD)	Time of day starting with TOD#0:0:0 in ms (UDINT)
DATE	Number of days since DATE#1992_01_01 (UDINT). DATE#1992_01_01 corresponds to 1
DATE_AND_TIME (DT)	Summary of TOD and DATE in the order specified.

Note

The marshalling function result may cause errors when the program is running, the error response set in the task configuration will then be triggered, see Execution errors in programs (Page 154).

Proceed with caution when converting byte arrays to the general ANY_REAL data type or to structures that contain this data type. The bit string from the byte array is taken unchecked as the ANY_REAL value. You must make sure that the bit string of the byte array corresponds to the bit pattern of a normalized floating-point number in accordance with IEEE 754. To do this, you can use the _finite (Page 408) and _isNaN (Page 409) functions or the IS_VALID (Page 410) function.

Otherwise, an error can be triggered (FPU exception (Page 155)) as soon as the ANY_REAL value is first used for an arithmetic operation (for example, in the program or when monitoring in the symbol browser).

Table 9-23 Example of marshalling function use

```
TYPE
    t_struct_1 : STRUCT
        m_word       : WORD;
        m_byte       : ARRAY [0..1] OF BYTE;
    END_STRUCT;
    t_struct_2 : STRUCT
        m_struct     : ARRAY [0..3] OF t_struct_1;
        m_lreal      : LREAL;
    END_STRUCT;
END_TYPE

VAR
    gsbVar           : t_struct_2;
    big_b_Array      : ARRAY [0..23] OF BYTE;
    lit_b_Array      : ARRAY [0..23] OF BYTE;
END_VAR

// Assignment of the values to the structure
gsbVar.m_struct[0].m_word     := WORD#16#7FF1;
gsbVar.m_struct[0].m_byte[0]  := BYTE#16#F9;
gsbVar.m_struct[0].m_byte[1]  := BYTE#16#E8;
gsbVar.m_struct[1].m_word     := WORD#16#9FF7;
gsbVar.m_struct[1].m_byte[0]  := BYTE#16#80;
gsbVar.m_struct[1].m_byte[1]  := BYTE#16#A1;
gsbVar.m_struct[2].m_word     := WORD#16#A881;
gsbVar.m_struct[2].m_byte[0]  := BYTE#16#BC;
gsbVar.m_struct[2].m_byte[1]  := BYTE#16#CD;
gsbVar.m_struct[3].m_word     := WORD#16#CF9C;
gsbVar.m_struct[3].m_byte[0]  := BYTE#16#D8;
gsbVar.m_struct[3].m_byte[1]  := BYTE#16#E7;
gsbVar.m_lreal                := LREAL#-12345.6789e123;
// Conversion to big endian
big_b_Array := AnyType_to_BigByteArray (
                    anyData  := gsbVar,
                    offset   := 0);
 // Content of the elements of big_b_array (big endian):
 // See 2nd column in the following table

// Conversion to little endian
lit_b_Array := AnyType_to_LittleByteArray (
                    anyData  := gsbVar,
                    offset   := 0);
 // Content of the elements of lit_b_array (little endian):
 // See 3rd column in the following table

// Conversion from big endian
gsbVar := BigByteArray_to_AnyType (
                    bytearray := big_b_Array,
                    offset    := 0);

// Conversion from little endian
gsbVar := LittleByteArray_to_AnyType (
                    bytearray := lit_b_Array,
                    offset    := 0);
```

Table 9-24 Content of the array elements of big_b_array and lit_b_array from example

Byte array big_b_array or lit_b_array			Components of the gsbVar variable	
Array index	**big_b_array (big endian)**	**lit_b_array (little endian)**	**Name**	**Value**
23	BYTE#16#07	BYTE#16#DA	m_lreal	LREAL#-12345.6789e123
22	BYTE#16#F0	BYTE#16#52		
21	BYTE#16#43	BYTE#16#3C		
20	BYTE#16#68	BYTE#16#EC		
19	BYTE#16#EC	BYTE#16#68		
18	BYTE#16#3C	BYTE#16#43		
17	BYTE#16#52	BYTE#16#F0		
16	BYTE#16#DA	BYTE#16#07		
15	BYTE#16#D8	BYTE#16#D8	m_struct[3].m_byte[1]	BYTE#16#D8
14	BYTE#16#E7	BYTE#16#E7	m_struct[3].m_byte[0]	BYTE#16#E7
13	BYTE#16#9C	BYTE#16#CF	m_struct[3].m_word	WORD#16#CF9C
12	BYTE#16#CF	BYTE#16#9C		
11	BYTE#16#CD	BYTE#16#CD	m_struct[2].m_byte[1]	BYTE#16#CD
10	BYTE#16#BC	BYTE#16#BC	m_struct[2].m_byte[0]	BYTE#16#BC
9	BYTE#16#81	BYTE#16#A8	m_struct[2].m_word	WORD#16#A881
8	BYTE#16#A8	BYTE#16#81		
7	BYTE#16#A1	BYTE#16#A1	m_struct[1].m_byte[1]	BYTE#16#A1
6	BYTE#16#80	BYTE#16#80	m_struct[1].m_byte[0]	BYTE#16#80
5	BYTE#16#F7	BYTE#16#9F	m_struct[1].m_word	WORD#16#9FF7
4	BYTE#16#9F	BYTE#16#F7		
3	BYTE#16#E9	BYTE#16#E9	m_struct[0].m_byte[1]	BYTE#16#E9
2	BYTE#16#F9	BYTE#16#F9	m_struct[0].m_byte[0]	BYTE#16#F9
1	BYTE#16#F1	BYTE#16#7F	m_struct[0].m_word	WORD#16#7FF1
0	BYTE#16#7F	BYTE#16#F1		

9.24.6 Communication functions

9.24.6.1 Available functions

ST provides the following functions for communication over non-configured connections:

- _Xsend, see Parameter description for _Xsend (Page 505)
- _GetStateOfXCommand, see Parameter description for _GetStateOfXCommand (Page 507)
- _Xreceive, see Parameter description for _Xreceive (Page 507)
- _tcpsend, see Communication via Ethernet with TCP/IP protocol (Page 511)
- _tcpreceive

- _udpsend, see Communication via Ethernet with UDP protocol (Page 512)
- _udpreceive

These communication functions are used to send and receive data:

- Between SIMOTION devices
- Between SIMOTION and SIMATIC S7 devices
  (S7-300, S7-400, M7-300, M7-400, etc.).
- Via acyclic communication; see also Acyclic communication with the drive (Page 512).

The dual functions **_Xsend** and **_Xreceive** enable transparent transmission of data packets that are sent coordinately by the user program of the client and received coordinately by the user program of the server.

Further information

Additional information is also available in the **Communication System/Configuration Manual** or in Acyclic communication with the drive (Page 512).

The program on the receiving device recognizes whether it is the data packet to be received on the basis of a user-definable integer appended to the data packet. The data exchange is only successful if the user program accepts and does not reject the data packet.

The sent data is in the form of byte sequences in an array, i.e. the data has no logical structure. SIMOTION devices can send or receive up to 200 bytes at once; the actual user data length depends on the communication peer.

You can check the status of an XSend or XReceive request with the **_GetStateOfXCommand** command.

9.24.6.2 Parameter description for _Xsend

The _Xsend function sends a data packet with transparent data to a communication peer. For the detailed syntax of the parameters, refer to the List Manual for the SIMOTION devices.

An overview is provided below:

- You can choose between two communication modes (*communicationMode* parameter): The connection is or is not retained after the data transmission.
- The *address* parameter specifies the destination address of the communications peer. The parameter is of data type *StructXSendDestAddr*. The following table lists the meaning of the individual components.

Table 9-25 Structure of destination address

Parameter (data type)	Meaning/values	Values	
deviceId (USINT)	Point of the connection	SIMOTION C230-2, C240, C240 PN	**1** for X8 **2** for X9
		SIMOTION P350, P350-3	**1** for X101 **2** for X102
		SIMOTION D410 DP	**1** for X21
		SIMOTION D410-2 DP	**2** for X21 **1** for X24
		SIMOTION D410-2 DP/PN	**2** for X21
		SIMOTION D4x5, D4x5-2	**1** for X126 **2** for X136
remoteSubnetIdLength (USINT)	Length of the subnet dialog box	**0** for MPI, PROFIBUS	
remoteStaddrLength (USINT)	Length of the station address (station number) of the destination system.	**1** for MPI, PROFIBUS	
nextStaddrLength (USINT)	Length of the router address	**0** for MPI, PROFIBUS	
remoteSubnetId (ARRAY [0..5] OF USINT)	Subnet mask	(Irrelevant for MPI, PROFIBUS)	
remoteStaddr (ARRAY [0..5] OF USINT)	Station address of the target system (actual destination address)	Station number for MPI, PROFIBUS, e.g. remoteStaddr[0] = 25	
nextStaddr (ARRAY [0..5] OF USINT)	Router address	(Irrelevant for MPI, PROFIBUS)	

Additional parameters of the _Xsend function are:

- The receiver identifies the data packet from the **job identifier** (*messageId* parameter) that you append to the data packet.
- You can select between two **modes of data transmission** (*nextCommand* parameter): synchronous or asynchronous.
  - During synchronous data transmission, the program waits until the receiver acknowledges receipt of the data packet before resuming execution. This happens automatically on receipt of the data packet.
  - During asynchronous data transmission, the program is resumed immediately after the command is issued. You can check the status of the command with *_GetStateOfXCommand*
- The mandatory **CommandId** parameter is used for internal command detection in ST. The parameter value should be saved to a local variable (data type *CommandIdType*) with the _getCommandId (Page 448) function. This variable can be used as a parameter value.
- The **data packet** (*data* parameter) involves a list containing 200 entries of 1 byte each. The list does not have to be this long if the amount of data you are sending is less than the maximum.

- The **data length** (*dataLength* parameter) is used to specify the actual length of the data packet to be transmitted.
- You can determine whether the command was successfully executed (return value = 0) on the basis of the **return value**.
  If a return value other than 0 is returned, an error has occurred (see the command syntax in the List Manual for the SIMOTION devices).

9.24.6.3 Parameter description for _Xreceive

The *_Xreceive* function is used to receive transparent data sent by a communication peer with *_Xsend*.

Below is a brief overview of the function parameters (see also the section on system functions in the List Manual for the SIMOTION devices):

- You identify the anticipated data packet from the **job identifier** (*messageId* parameter) appended by the sender.
- As when sending data, you can choose between two **modes of data reception** (*nextCommand* parameter): synchronous or asynchronous.
  - During synchronous data receipt, the program waits until the data packet has arrived before it resumes execution. Receipt of the data packet is then acknowledged automatically to the sender.
  - During asynchronous data transmission, the program is resumed immediately after the command is issued. You can check the status of the commands with *_GetStateOfXCommand*.
- The mandatory **CommandId** parameter is used for internal command detection in ST. The parameter value should be saved to a local variable (data type *CommandIdType*) with the _getCommandId (Page 448) function. This variable can be used as a parameter value.
- The **return value** is a structure:
  - You can determine whether the command was successfully executed (functionResult = 0) on the basis of the *functionResult* element.
    If a value other than 0 is returned, an error has occurred (see the command syntax in the List Manual for the SIMOTION devices).
  - The *dataLength* element represents the data length of the data packet received.
  - The *data* element represents the data packet received (array of up to 200 entries of one byte each).

9.24.6.4 Parameter description for _GetStateOfXCommand

You can check the status of an *_Xsend* or *_Xreceive* command with the _GetStateOfXCommand function.

Below is a brief overview of function parameters (see also documentation for the technology functions):

- The status query can tell which command is involved by the mandatory *commandId* parameter uniquely assigned to each send and receive function (see information about this parameter for *_Xsend* and *_Xreceive* above).
- The return value consists of an error number (zero when command execution is okay, otherwise greater than zero) and the status of the command being checked (see command syntax in the system functions for the SIMOTION devices).

9.24.6.5 Communication between SIMOTION and SIMATIC S7 devices

You must consider the following for communication between SIMOTION and SIMATIC S7 devices:

- Communication is only possible with *_Xsend* and *_Xreceive*.
- The maximum volume of data that can be transmitted in one packet is limited to 76 bytes. If you transmit larger data packets, an error message is output.
- The SIMOTION interface must be connected to the **MPI interface** of the SIMATIC S7 devices. The baud rate on the SIMOTION interface must be set to correspond to the baud rate of the SIMATIC S7device. For example, the baud rate for an SIMATIC S7-300 must be configured to 187.5 Kbits/s (see documentation for the relevant SIMATIC S7 devices).

Note

The parameters for the *_Xsend* and *_Xreceive* commands have different names in the SIMOTION system are in some cases have a different meaning than those you used in your previous SIMATIC S7 system. You will find a comparison in the following tables.
The *_GetStateOfXCommand* command does not exist in an SIMATIC S7 system, thus eliminating the need for comparison.

Table 9-26 Comparison of the parameters for _Xsend in SIMATIC S7 and SIMOTION devices

SIMATIC S7 device (parameters for SFC 65 X_SEND)	SIMOTION device (parameters for _Xsend)
REQ (Request to activate, data type BOOL)	– (Not available)
DEST_ID (MPI station number of the communications peer)	address (Destination address of the communication peer as a structure of data type StructXSendDestAddr)
CONT (Keep connected, Boolean value TRUE or FALSE)	communicationMode (Keep connected, enum. value M_TRUE or M_FALSE)
– (Not available)	nextCommand (Data transmission mode, enum values for synchronous and asynchronous)
REQ_ID (Job identifier, value of data type DWORD)	messageId (Job identifier, value of data type UDINT)

SIMATIC S7 device (parameters for SFC 65 X_SEND)	**SIMOTION device (parameters for _Xsend)**
SD (Variable address for sent data, value of data type ANY pointer, max. 76 bytes long)	data (Data packet to be sent, one-dim. array with max. 200 values of data type USINT)
– (Not available)	dataLength (Data length of data packet to be sent in bytes, value of data type UDINT)
– (Not available)	commandId (Internal command identifier, see description in parameter description above)
BUSY (Activation status, values of data type BOOL)	– (Not available)
<Return value> (= 0, if no error occurred; <> 0, if error occurred; see SIMATIC S7 documentation)	<Return value, (= 0, if no error occurred; <> 0, if error occurred; see Parameter Manual for the SIMOTION devices)

Table 9-27 Comparison of the parameters for _Xreceive in SIMATIC S7 and SIMOTION devices

SIMATIC S7 device (parameters for SFC 66 X_RCV)	**SIMOTION device (parameters for _Xreceive)**
EN_DT (Enable Data Transfer, data type BOOL)	– (Not available)
REQ_ID (Job identifier, value of data type DWORD)	messageId (Job identifier, value of data type UDINT)
– (Not available)	nextCommand (Data transmission mode, enum values for synchronous and asynchronous)
– (Not available)	commandId (Internal command identifier, see description in parameter description above)
RD (variable address for received data, value of data type ANY pointer, max. 76 bytes long)	<Return value, data structure element> (Received data packet; an array of up to 200 entries of 1 byte each)
– (Not available)	<Return value, dataLength structure element> (Data length of received data package)
<Return value> (= 0, if no error occurred; <> 0, if error occurred; see SIMATIC S7 documentation)	<Return value, functionResult structure element> (= 0, if no error occurred; <> 0, if error occurred; see Parameter Manual for the SIMOTION devices)

9.24.6.6 Example of send and receive program

The following figures show the source text for the send and receive program.

Table 9-28 Example of send program

```
INTERFACE
    PROGRAM xsend_control;
END_INTERFACE

IMPLEMENTATION
                          // The following program must be assigned
                          // to a MotionTask.
                          // In the task configuration, the "Activation
                          // after Startup Task" option must be selected.
PROGRAM xsend_control
VAR
    retVal                          : DINT;
    myAddress                       : StructXSendDestAddr;
    myStaddr                        : ARRAY[0..4] OF BYTE;
    myData                          : ARRAY[0..199] OF BYTE;
END_VAR
                          // Specify destination address and PROFIBUS interface ////////
myAddress.deviceID                  := 1;
                          // PROFIBUS interface X8
myAddress.remoteStaddrLength        := 1;
                          // must always be assigned a value of 1
myAddress.remoteStaddr[0]           := 4;
                          // PROFIBUS address of the receiving station

                          // Send data
myData[0] := 170;

                          // Call of send function
retVal := _Xsend( communicationMode := ABORT_CONNECTION
             , address              := myAddress
             , messageId            := 1
             , nextCommand          := WHEN_COMMAND_DONE
             , commandId            := _getCommandId()
             , data                 := myData
             , dataLength           := 1
             );
END_PROGRAM
END_IMPLEMENTATION
```

Table 9-29 Example of receive program

```
INTERFACE
    PROGRAM xreceive_control;
END_INTERFACE

IMPLEMENTATION
VAR_GLOBAL
    retVal                          : StructRetXReceive;
END_VAR
                          // The following program must be assigned
                          // to a MotionTask.
                          // In the task configuration, the "Activation
                          // after Startup Task" option must be selected.
PROGRAM xreceive_control

// Call of receive function
retVal := _Xreceive( messageId       := 1
            , nextCommand            := WHEN_COMMAND_DONE
            , commandId              := _getCommandId()
            );
END_PROGRAM
END_IMPLEMENTATION
```

9.24.6.7 Communication via Ethernet with TCP/IP protocol

For SIMOTION devices with Ethernet interface, communication via the TCP/IP protocol is also possible.

The following table lists the individual steps to establish communication between a sender (client) and a receiver (server):

Table 9-30 Communication steps of a TCP/IP connection and the corresponding system functions

Communication steps		System function
Establish the connection		
1	Receiver (server) waits for communication request.	_tcpOpenServer
2	Sender (client) requests connection establishment to the receiver.	_tcpOpenClient
3	Receiver (server) has established communication request.	
Communicating		
4	Sender sends data to the receiver.	_tcpSend
5	Receiver receives data from the sender.	_tcpReceive
Terminating communication connection		
6	Sender does not send any more data and closes the connection.	_tcpCloseConnection
7	There is no further connection required.	_tcpCloseServer

The system functions are described in detail in the List Manual for the SIMOTION devices, in the chapter on system functions.

Note

A sender or receiver can be a client as well as a server when establishing a connection.

There must be at least one client and one server to establish TCP/IP connection.

The client-server relationship is only valid until the connection is established. After connection is established, both communication partners are equivalent, i.e. each of the two can send or receive or close the connection at any point of time.

For a detailed description, refer to the *Frequently Asked Questions* (FAQs) on *SIMOTION Utilities & Applications* (DVD).

9.24.6.8 Communication via Ethernet with UDP protocol

For SIMOTION devices with Ethernet interface, communication via the UDP protocol is also possible.

- The *_udpSend* function sends a data packet to a communication partner which is specified with IP address and port number.
  The data to be sent is transferred to the function as ARRAY [0..1399] OF BYTE.
- The *_udpReceive* function is used to receive a data packet which a communication partner has sent with *_udpSend*.
  The received data is stored in a variable of data type ARRAY [0..1399] OF BYTE, whose identifier is transferred to the function as parameter.

The system functions are described in detail in the List Manual for the SIMOTION devices, in the chapter on system functions.

A detailed description is available in the FAQs on *SIMOTION Utilities & Applications* (DVD).

9.24.6.9 Acyclic communication with the drive

Overview

PROFIdrive drive units are supplied with control signals and setpoints by the controller and return status signals and actual values. These signals are normally transferred cyclically (i.e. continuously) between the controller and the drive.

For SINAMICS S110/S120, configure the axis message frames for data exchange (see Setting up addresses and message frames - overview (Page 109)).

As well as offering cyclic data exchange, PROFIdrive drive units have an acyclic communication channel. In particular, this is used for reading and writing drive parameters (e.g. error codes, warnings, controller parameters, motor data, etc.).

As a result, data can be transferred on an "acyclic" as opposed to a "cyclic" basis when required. Acyclic reading and writing of parameters for PROFIdrive drives is based on the DP V1 services "Read data set" and "Write data set".

The acyclic DP V1 services are transferred in parallel to the cyclic communication via PROFIBUS or PROFINET. The PROFIdrive profile specifies precisely how these basic mechanisms are used for read/write access to parameters of a PROFIdrive-compliant drive.

The PROFIdrive standard states that "pipelining" of jobs on PROFIdrive drives is not supported. This means:

- Only one "Write/read data set" can be performed at any one time on a drive unit (e.g. SINAMICS S110/S120 control unit or the SINAMICS Integrated of a SIMOTION D).
- However, if several PROFIdrive drive units are connected to a controller, a job can be processed for each of these drive units at the same time. In this case, the maximum total number of jobs will depend on the controller (maximum of eight jobs simultaneously with SIMOTION).

For acyclic data exchange with SINAMICS drives, this means you will have to coordinate the write/read jobs with each other (buffer management). An interlock must be set to prevent the application or different parts of the application from sending overlapping jobs to the same PROFIdrive drive unit.

Additional references

Additional information on how to use DP V1 services can be found in the *SIMOTION Communication* System Manual.

SIMOTION Utilities & Applications also has a DP V1 library with functions that are capable of performing coordination tasks commonly associated with acyclic communication. The library not only coordinates access to the system functions (_ReadRecord/_WriteRecord/ _readDriveParameter/_writeDriveParameter/, etc.), but also expands the range of functions for frequently required tasks, e.g. the reading of faults and alarms from the drive unit.

SIMOTION Utilities & Applications is part of the scope of delivery of SIMOTION SCOUT.

The following functions are available in the DP V1 library:

- Buffer management (coordination of a number of parallel DP V1 services)
- StartUp (function for coordinating the power-up of the SINAMICS drive with SIMOTION)
- TimeSync (applicative time synchronization: Transfer of SIMOTION time of day to the SINAMICS drives)
- SetActIn (activating/deactivating objects in SIMOTION and SINAMICS)
- RwnPar (reading and writing of drive parameters)
- GetFault (reading errors and warnings from the drive)

See also

Available functions (Page 504)

9.24.7 Synchronous start

On synchronous start, several commands are started within an IPO cycle clock or IPO_2 cycle clock.

Follow these steps:

1. First, obtain a unique SyncCommandId from the system. You will need this SyncCommandId as a unique designation for the synchronized commands.
   For this purpose, use the _getSyncCommandId() system function.
2. Enclose the commands to be executed synchronously between the BEGIN_SYNC(SyncCommandId) and END_SYNC() functions. This defines them for synchronized start.
   All motion commands are permissible within the structure BEGIN_SYNC/END_SYNC. The value IMMEDIATELY must be transferred as the nextCommand parameter.
3. The synchronous start itself occurs with the _startSyncCommand (SyncCommandId) function. The commands defined for the synchronized start are processed in parallel.

See sample program.

Note

Synchronous start for motion commands is only ensured if the following conditions are satisfied:

1. Commands included in a synchronous start must act on various technology objects.
2. The technology objects involved must be in the same execution level (IPO or IPO_2).
3. When you use the Synchronous Start command in MCC, you must generate the UserInterruptTask_1 since this is called in the event of an error. You can program an error response in this task, see UserInterruptTasks (Page 237).

The task controller is temporarily shut down during the synchronous start. It is not restarted until the following conditions are met:

- All single-axis commands and synchronous operation and cam commands in the branches have been started
- All basic commands in the branches have been completed

Start interruptions by other tasks (except SynchronousTasks) are thus prevented. This can cause a timeout to occur in cyclic tasks (BackgroundTask, TimerInterruptTasks). This error can be detected and caught by programming the TimeFaultBackgroundTask or TimeFaultTask appropriately.

The BEGIN_SYNC, END_SYNC, and *_startSyncCommand* functions are system functions of SIMOTION devices; for more information, refer to the parameter manual for the relevant SIMOTION device.

Note

Synchronous start is often used in conjunction with the WAITFORCONDITION construct. In this case, the system waits for a condition to be fulfilled before the synchronized start. The *_startSyncCommand* function must not occur within the WAITFORCONDITION construct.

Table 9-31 Sample program for synchronized start of two axes with WAITFORCONDITION

```
INTERFACE
    USEPACKAGE cam;
    PROGRAM sync_motion;
END_INTERFACE

IMPLEMENTATION

    EXPRESSION wait_sync_expression
        wait_sync_expression := TRUE;
    END_EXPRESSION

    PROGRAM sync_motion
        VAR
            ret_val : DINT;
            sync_id : CommandIdType;
        END_VAR

        sync_id := _getSyncCommandId();
        BEGIN_SYNC(sync_id);
        (* Positioning axis ('Pos') *)
        ret_val := _pos    (axis := Axis_1,
                    positioningMode := ABSOLUTE,
                    position := 100,
                    mergeMode := IMMEDIATELY,
                    nextCommand := IMMEDIATELY,
                    commandId := _getCommandId() );
        (* Positioning axis ('Pos') *)
        ret_val := _pos    (axis := Axis_2,
                    positioningMode := ABSOLUTE
                    position := 50
                    mergeMode := IMMEDIATELY
                    nextCommand := IMMEDIATELY,
                    commandId := _getCommandId() );
        END_SYNC();

        WAITFORCONDITION wait_sync_expression DO
            ;
        END_WAITFORCONDITION;

        ret_val := _startSyncCommands(sync_id);
        // Here in the event of an error Start of the UserInterruptTask
        IF (_ret_val <> 0) THEN
            _startTaskId(_task.UserInterruptTask_1);
        END_IF;

    END_PROGRAM
END_IMPLEMENTATION
```

To check the correct completion of the synchronous start, you can query whether the commands of the two axes started in the previous example have been completed without error.

The following MCC sample program shows a synchronous start with two axes and subsequent query of the group variable _MccRetSyncStart. This is not equal to 0 when a command within the synchronous start had a return value not equal to 0.

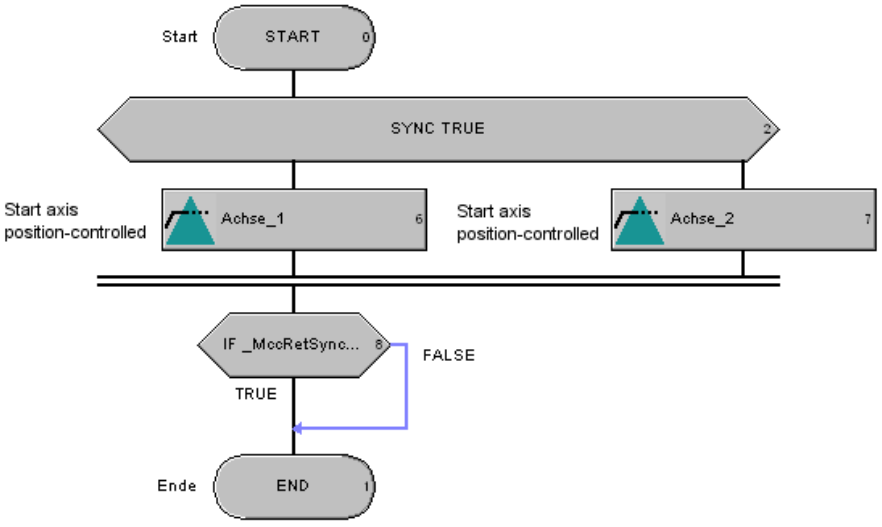


Figure 9-1 MCC synchronous start

The following is the section from the appropriate ST program (generated here via export from MCC unit)

Table 9-32 ST synchronous start

```
(* Synchronous start ('StartSync') *)
_MccCommand1 := _getCommandId();
_MccCommand2 := _getCommandId();

_MccSync := _getSyncCommandId();
BEGIN_SYNC(_MccSync);
    (* Start axis position-controlled ('Move') *)
    _MccRetDINT_1:=_move(axis:=Achse_1, velocityType:=DIRECT,
        velocity:=v_blue, moveTimeOutType:=WITHOUT_TIME_LIMIT,
        mergeMode:=IMMEDIATELY, nextCommand:=IMMEDIATELY,
        commandId:=_MccCommand1, movingMode:=POSITION_CONTROLLED);
    (* Start axis position-controlled ('Move') *)
    _MccRetDINT_2:=_move(axis:=Achse_2, velocityType:=DIRECT,
        velocity:=v_red, moveTimeOutType:=WITHOUT_TIME_LIMIT,
        mergeMode:=IMMEDIATELY, nextCommand:=IMMEDIATELY,
        ommandId:=_MccCommand2, movingMode:=POSITION_CONTROLLED);
(* Synchronous start ('EndSync') *)
END_SYNC();

WAITFORCONDITION _MccMCC_1Condition1 DO
    ;
END_WAITFORCONDITION;

_MccRetDINT := _startSyncCommands(_MccSync);
// Start of the UserInterruptTask
IF (_MccRetDINT <> 0) THEN
    _startTaskId(_task.UserInterruptTask_1);
END_IF;

_MccRetStructRetMotionCommandState := _getMotionStateOfAxisCommand
    (Achse_1, _MccCommand1);

// Query whether both axis commands are completed

WHILE ((_MccRetStructRetMotionCommandState.functionResult = 0 )
   AND (_MccRetStructRetMotionCommandState.motionCommandIdState <>
   IN_CONSTANT_MOTION) AND
   (_MccRetStructRetMotionCommandState.motionCommandIdState <>
    NOT_EXISTENT)) DO
    _MccRetDINT := _waitTime(T#0d_0h_0m_0s_0ms);
    _MccRetStructRetMotionCommandState :=
    _getMotionStateOfAxisCommand(Achse_1, _MccCommand1);
END_WHILE;

_MccRetStructRetMotionCommandState :=
_getMotionStateOfAxisCommand(Achse_2, _MccCommand2);

WHILE ((_MccRetStructRetMotionCommandState.functionResult = 0 )
   AND (_MccRetStructRetMotionCommandState.motionCommandIdState <>
   IN_CONSTANT_MOTION) AND
   (_MccRetStructRetMotionCommandState.motionCommandIdState <>
   NOT_EXISTENT)) DO
   _MccRetDINT := _waitTime(T#0d_0h_0m_0s_0ms);
   _MccRetStructRetMotionCommandState :=
```

```
   _getMotionStateOfAxisCommand(Achse_2, _MccCommand2);
END_WHILE;

_MccRetSyncStart := 0; // Query whether both axis commands are running without error
IF (_MccRetDINT_1 + _MccRetDINT_2 <> 0) THEN
    _MccRetSyncStart := -1;
END_IF;

(* IF: Program branch ('If') *)
IF (_MccRetSyncStart <> 0) THEN
    ;// Error handling
(*  ('Else') *)
ELSE
    ;
(*  ('EndIf') *)
END_IF;
```

9.25 HMI (Human Machine Interface) connection

9.25.1 Interface between HMI and SCOUT or SIMOTION

SIMOTION allows communication with HMI devices (operator devices for the end user), e.g. with operator panels.

If one or more SIMOTION devices are connected on the PROFIBUS or PROFINET, the HMI device can display variables, messages and alarms of the SIMOTION devices. Likewise, you can initiate functions programmed using the HMI device and a program in the SIMOTION device.

The WinCC flexible software package is available for implementing such tasks on the HMI device. You can use WinCC flexible to read and write to system variables and unit variables declared in the interface section. See also HMI variables in one unit (Page 602).

Note

If you would like to use a SIMOTION module as DP slave on an HMI, you must make the following settings in HW Config for the object properties of the relevant DP interface:

1. Select the **Operating mode** tab.
2. Select **DP Slave** as the setting.
3. Activate the Option **Programming, status/control or other PG functions and non-configured communication connection possible**.

SIMOTION also provides an OPC server. You can use this server to access process data with non-proprietary HMI software. The configuration software for WinCC flexible allows direct access to the symbolic variables of a SIMOTION device. Likewise, this is possible via OPC. For

this purpose, the configuration data of WinCC flexible must be located in the same project as the SIMOTION devices.

Note

- Variables that are meant to be available in HMI must always be created as unit variables in the interface section of a source.
- HMI tags can only be linked to SIMOTION variables of a simple data type (e.g. DINT). Other variables such as ANYOBJECT or configuration data cannot be linked.

Further information about WinCC flexible

Notes on the configuration with WinCC flexible can be found:

- In the online help for WinCC flexible
- In the online help for SCOUT when WinCC flexible is integrated in SCOUT

Further FAQs are also available online at:

http://support.automation.siemens.com/WW/view/en/28767398

http://support.automation.siemens.com/WW/view/en/23751257

Additional information on the topics below is available in the FAQs in SIMOTION Utilities & Applications:

- Time synchronization SIMOTION
- WinCC/WinCCFlexible
- SIMOTION and HMI (parallel work on SIMOTION and HMI projects)

Examples of HMI configurations can be found among the standard applications contained within the Applications section on SIMOTION Utilities & Applications.

- Examples of HMI implementation

See also HMI variables in one unit (Page 602).

Alarms

Everything that is displayed in the alarm list in SCOUT can also be displayed in WinCC flexible with the appropriate configuration, except SINAMICS alarms!

All SIMOTION and SINAMICS alarms function with OPC Alarm&Event.

The present mechanism is as follows:

When the HMI project is generated, the SCOUT texts (axis names, axis alarms, Alarm_S, etc.) are synchronized with the HMI, whereby each technology object has a number.

Therefore, when an alarm occurs the TO number, alarm number, coefficient, etc. are signaled and the HMI then writes the TO number, alarm text with coefficient, time stamp, etc.

With WinCC flexible, warning and error messages of the technology objects (axes, output cams, measuring inputs, etc.) are output directly on the HMI and are acknowledged by the operator. The AlarmS concept also provides a method for configuring and programming user messages. User messages are configured in SIMOTION SCOUT; messages are called and

acknowledged during runtime by calling the appropriate system commands. OPC supports the same message mechanisms. For the OPC export, an additional OPC alarm/event must be activated.

9.25.2 Consistent data access with HMI devices (example)

HMI devices can also access user data of the SIMOTION device consistently. The following example includes an ST program on the SIMOTION device (see example) and an application (Visual Basic program) on the HMI device (see example).

The user requests consistent read access to user variables (including arrays) from within the HMI application (in this case, Visual Basic; but it could also be Visual C++ ...). An uneven value is written to the consistencyFlag variable. The SIMOTION user program then copies the data. The HMI application waits until the SIMOTION user program confirms that it has completed copying the data by writing an even value to the *consistencyFlag* variable. Then, the HMI reads out this variable consistently, since no changes have been made in the meantime.

The ST program can be assigned to a cyclic task; it may be necessary to take the IPOsynchronousTask if, for example, axis positions and velocities are to be read at the same time (as a snapshot).

Examples:

Table 9-33 ST program for consistent data access from HMI devices

```
INTERFACE
    VAR_GLOBAL
        consistencyFlag : DINT;
        myDint    : DINT;
        myArray : ARRAY[0..10] OF LREAL;
    END_VAR
    PROGRAM OPC_Prog;
END_INTERFACE

IMPLEMENTATION
    PROGRAM OPC_Prog
        IF (consistencyFlag MOD 2) = 1 THEN
            myDint := 99;
            myArray[0] := 0.0;
            myArray[1] := 1.0;
            myArray[10] := 10.0;
            consistencyFlag := consistencyFlag + 1;
        END_IF;
    END_PROGRAM
END_IMPLEMENTATION
```

Table 9-34 HMI application for consistent data access (Visual Basic)

```
Option Explicit
Option Base 1

    Dim g_Server As OPCServer
    Dim g_GroupObj As OPCGroup

    Dim g_myItem1 As OPCItem
    Dim g_myItem2 As OPCItem

    Dim g_myItem3 As OPCItem
    Const OPC_DS_DEVICE = 2

    Dim consistencyFlag As Long

Private Sub Form_Load()
    Set g_Server = New OPCServer
    g_Server.Connect ("OPC.SimaticNet")
    Set g_GroupObj = g_Server.OPCGroups.Add("Test1")
    g_GroupObj.IsActive = False
    Set g_myItem1 = g_GroupObj.OPCItems.AddItem("C240.consistencyFlag", 2)
    Set g_myItem2 = g_GroupObj.OPCItems.AddItem("C240.myDint", 2)
    Set g_myItem3 = g_GroupObj.OPCItems.AddItem("C250.myArray", 3)
    consistencyFlag = 1
End Sub
Private Sub Form_Unload(Cancel As Integer)
    Set g_myItem1 = Nothing
    Set g_myItem2 = Nothing
    Set g_myItem3 = Nothing
    Set g_GroupObj = Nothing
    Set g_Server = Nothing
    End
End Sub

Private Sub cmdRead_Click()
    Static reentrancyFlag As Boolean
    Dim var1 As Variant
    Dim var2 As Variant
    Dim var3 As Variant

    If reentrancyFlag = False Then
        reentrancyFlag = True
        consistencyFlag = consistencyFlag + 2
        g_myItem1.Write consistencyFlag
    Do
        g_myItem1.Read OPC_DS_DEVICE, var1
    Loop Until var1 = consistencyFlag + 1
        g_myItem2.Read OPC_DS_DEVICE, var2
        g_myItem3.Read OPC_DS_DEVICE, var3
        reentrancyFlag = False
    End if
End Sub
```

Programming of general system function blocks 10

10.1 Overview of the function blocks

ST contains a series of system function blocks that you can use in your ST source files without having to declare them beforehand. You only have to create an instance and assign the necessary parameters.

Overview of system function blocks

Below is an overview of all system function blocks implemented in the system. You will find definitions and a further specification in the following sections.

Table 10-1 System function blocks

Collective term	Name	Input parameter		Output parameter	
	Definition	Name	: Type	Name	: Type
Bistable function blocks (Page 525)	**SR**[1] Priority set	S1 R	: BOOL; : BOOL;	Q1	: BOOL;
	RS[1] Priority reset	SR1	: BOOL; : BOOL;	Q1	: BOOL;
Edge detection (Page 528)	**R_TRIG**[1] Rising edge	CLK	: BOOL;	Q	: BOOL;
	F_TRIG[1] Falling edge	CLK	: BOOL;	Q	: BOOL;

Collective term	Name	Input parameter		Output parameter	
	Definition	**Name**	**: Type**	**Name**	**: Type**
Counters (Page 530)	**CTU**[1]	CU	: BOOL;	Q	: BOOL;
	Up counter	R	: BOOL;	CV	: INT;
	Data type: INT	PV	: INT;		
	CTU_DINT[1]	CU	: BOOL;	Q	: BOOL;
	Up counter	R	: BOOL;	CV	: DINT;
	Data type: DINT	PV	: DINT;		
	CTU_UDINT[1]	CU	: BOOL;	Q	: BOOL;
	Up counter	R	: BOOL;	CV	: UDINT;
	Data type: UDINT	PV	: UDINT;		
	CTD[1]	CD	: BOOL;	Q	: BOOL;
	Down counter	LD	: BOOL;	CV	: INT;
	Data type: INT	PV	: INT;		
	CTD_DINT[1]	CD	: BOOL;	Q	: BOOL;
	Down counter	LD	: BOOL;	CV	: DINT;
	Data type: DINT	PV	: DINT;		
	CTD_UDINT[1]	CD	: BOOL;	Q	: BOOL;
	Down counter	LD	: BOOL;	CV	: UDINT;
	Data type: UDINT	PV	: UDINT;		
	CTUD[1]	CU	: BOOL;	QU	: BOOL;
	Up/down counter	CD	: BOOL;	QD	: BOOL;
	Data type: INT	R	: BOOL;	CV	: INT;
		LD	: BOOL;		
		PV	: INT;		
	CTUD_DINT[1]	CU	: BOOL;	QU	: BOOL;
	Up/down counter	CD	: BOOL;	QD	: BOOL;
	Data type: DINT	R	: BOOL;	CV	: UINT;
		LD	: BOOL;		
		PV	: DINT;		
	CTUD_UDINT[1]	CU	: BOOL;	QU	: BOOL;
	Up/down counter	CD	: BOOL;	QD	: BOOL;
	Data type: UDINT	R	: BOOL;	CV	: UDINT;
		LD	: BOOL;		
		PV	: UDINT;		
Timers (Page 535)	**TP**[1]	IN	: BOOL;	Q	: BOOL;
	Pulse	PT	: TIME;	ET	: TIME;
	TON[1]	IN	: BOOL;	Q	: BOOL;
	ON delay	PT	: TIME;	ET	: TIME;
	TOF[1]	IN	: BOOL;	Q	: BOOL;
	ON delay	PT	: TIME;	ET	: TIME;
	RTC[1]	SET	: BOOL;	CDT	: DT;
	Real-time clock	READ	: BOOL;		
		PDT	: DT;		

Collective term	Name	Input parameter		Output parameter	
	Definition	Name	: Type	Name	: Type
Splitting bit-string data types (Page 539)	**_BYTE_TO_8BOOL**	n	: BYTE	bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7	: BOOL;
	_WORD_TO_2BYTE	in	: WORD	byte0, byte1	: BYTE;
	_DWORD_TO_2WORD	in	: DWORD	word0, word1	: WORD;
	_DWORD_TO_4BYTE	in	: DWORD	byte0, byte1, byte2, byte3	: DWORD;
Emulation of SIMATIC S7 commands (Page 542)	**_S7_COUNTER**[1]	CU CD PV S R	: BOOL; : BOOL; : INT; : BOOL; : BOOL;	Q CV	: BOOL; : INT;
	_S7_TIMER[1]	T_Type PR S R TV TV_BCD	: WORD; : BOOL; : BOOL; : BOOL; : TIME; : WORD;	Q BI	: BOOL; : TIME;
[1] These function blocks require a repeated call (e.g. in a cyclic task).					

Comparison of the system function blocks for SIMOTION and SIMATIC

You can find a comparison of the SIMATIC S7 and SIMOTION system function blocks in the 2_FAQ directory on the Utilities & Applications CD.

10.2 Bistable elements (set flip-flop)

Description

In ST you can use system function block SR to set/reset the flip-flop (priority set) and system function block RS to reset/set the flip-flop (priority reset).

Bistable function block SR (priority set)

The stored value can be obtained at output Q1. Output Q1 is 1 when S1 is 1. If S1 and R are equal to 0, output Q1 does not change.

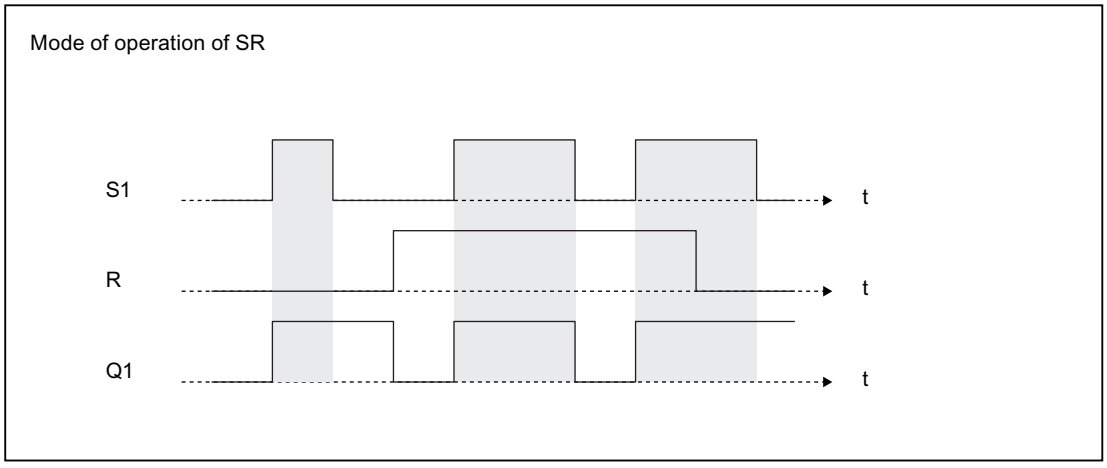


Figure 10-1 Mode of operation of the SR bistable function block

Program code of the SR bistable function block

```
FUNCTION_BLOCK SR

    VAR_INPUT
        S1, R    : BOOL;
    END_VAR

    VAR_OUTPUT
        Q1       : BOOL;
    END_VAR

    Q1 := S1 OR (NOT R AND Q1);

END_FUNCTION_BLOCK
```

Table 10-2 Call parameters for SR

Identifier	Parameter	Data type	Description
S1	Input	BOOL	Set
R	Input	BOOL	Reset
Q1	Output	BOOL	Signal state

Bistable function block RS (priority reset)

The stored value can be obtained at output Q1. Output Q1 is 0 when R1 is 1. If R1 and S are equal to 0, output Q1 does not change.

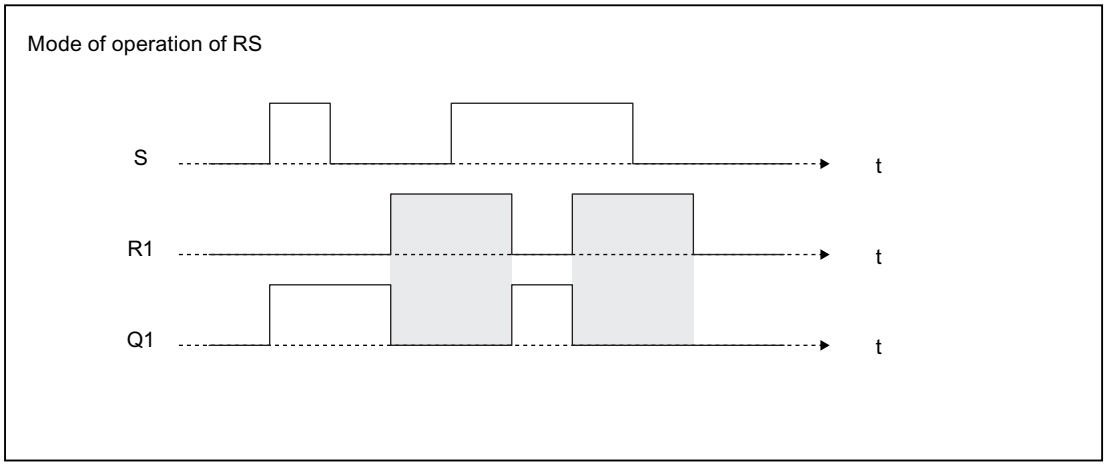


Figure 10-2 Mode of operation of the RS bistable function block

Program code of RS bistable function block

```
FUNCTION_BLOCK RS

    VAR_INPUT
        R1, S      : BOOL;
    END_VAR

    VAR_OUTPUT
        Q1         : BOOL;
    END_VAR

    Q1 := NOT R1 AND (S OR Q1);

END_FUNCTION_BLOCK
```

Table 10-3 Call parameters for RS

Identifier	Parameter	Data type	Description
S	Input	BOOL	Set
R1	Input	BOOL	Reset
Q1	Output	BOOL	Signal state

10.3 Edge detection

Description

System function block R_TRIG can be used to detect a rising edge; F_TRIG can detect a falling edge. You can use this function, for example, to set up a sequence of your own function blocks.

Detection of rising edge R_TRIG

If a rising edge (R_TRIG, Rising Trigger), i.e. a status change from 0 to 1 is present at the input, 1 is applied at the output for the duration of one cycle.

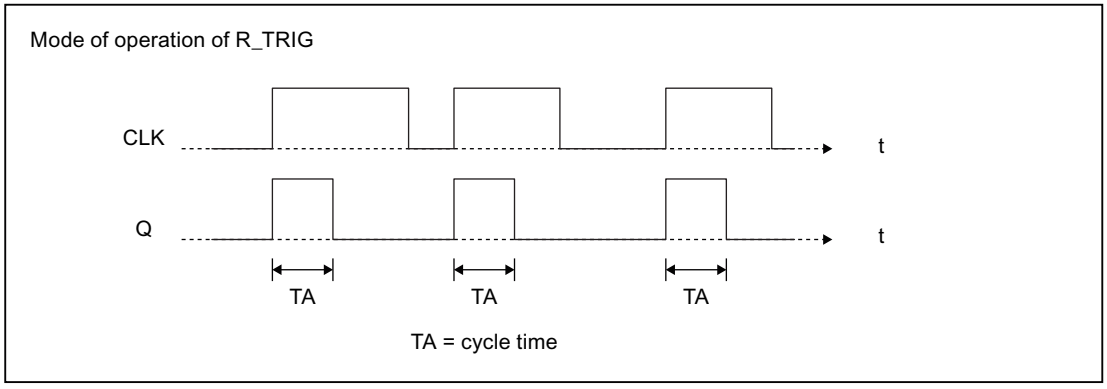


Figure 10-3 Mode of operation of R_TRIG (rising edge) function block

Program code for function block R_TRIG (rising edge)

```
FUNCTION_BLOCK R_TRIG

    VAR_INPUT
        CLK    : BOOL;
    END_VAR

    VAR_OUTPUT
        Q      : BOOL;
    END_VAR

    VAR
        M      : BOOL := 0;
    END_VAR

    Q := CLK AND NOT M;
    M := CLK;

END_FUNCTION_BLOCK
```

Table 10-4 Call parameters for R_TRIG

Identifier	Parameter	Data type	Description
CLK	Input	BOOL	Input for edge detection
Q	Output	BOOL	Status of edge

Detection of falling edge F_TRIG

When a falling edge (F_TRIG, falling trigger), i.e. a status change from 1 to 0, occurs at the input, the output is set to 1 for the duration of one cycle time.

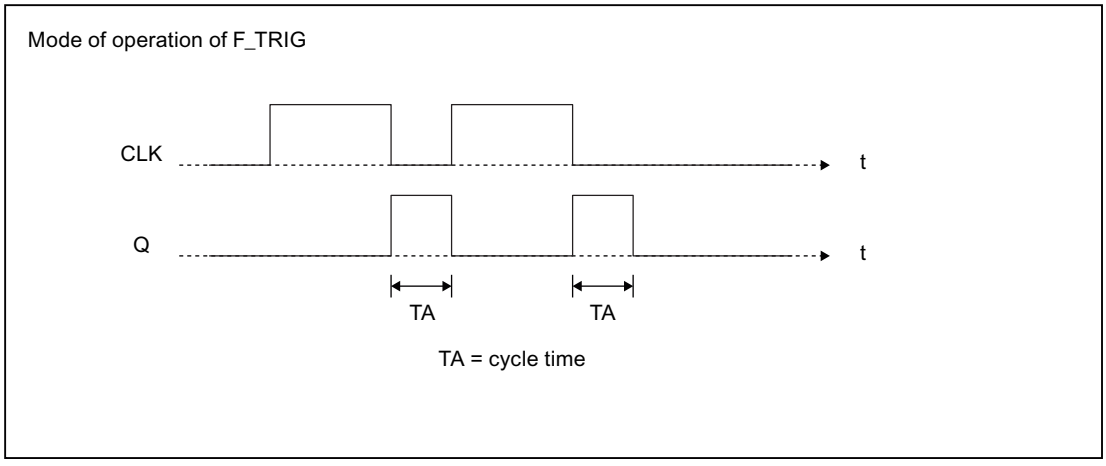


Figure 10-4 Mode of operation of F_TRIG (falling edge) function block

Program code for function block F_TRIG (falling edge)

```
FUNCTION_BLOCK F_TRIG

    VAR_INPUT
        CLK  : BOOL;
    END_VAR

    VAR_OUTPUT
        Q    : BOOL;
    END_VAR

    VAR
        M    : BOOL    := 1;
    END_VAR

    Q := NOT CLK AND NOT M;
    M := NOT CLK;

END_FUNCTION_BLOCK
```

Table 10-5 Call parameters for F_TRIG

Identifier	Parameter	Data type	Description
CLK	Input	BOOL	Input for edge detection
Q	Output	BOOL	Status of edge

10.4 Counters

10.4.1 General information on counters

ST provides a series of system function blocks for counters that you can use in your ST program.

10.4.2 CTU up counter

Description

The CTU counter allows you to perform upward counting operations:

- If the input is R = TRUE when the FB is called up, then the CV output is reset to 0.
- If the CU input changes from FALSE to TRUE (0 to 1) when the FB is called (rising edge), then the CV output is incremented by 1.
- Output Q specifies whether CV is greater than or equal to comparison value PV.

The CV and PV parameters are both INT data types, which means that the maximum counter value possible is 32_767 (= 16#7FFF).

User interface prototype

```
FUNCTION_BLOCK CTU
    VAR_INPUT
        CU    : BOOL;       // Count:
        R     : BOOL;       // Reset
        PV    : INT;        // Comparison value
    END_VAR

    VAR_OUTPUT
        Q     : BOOL;       // Status
        CV    : INT;        // Counter value
    END_VAR

    // ... (Code)

END_FUNCTION_BLOCK
```

Input parameter

CU

Data type: BOOL

Count up if value changes from FALSE to TRUE (rising edge)

R

Data type: BOOL

TRUE: Reset the counter to 0

PV

Data type: INT

Comparison value

Output parameter

Q

Data type: BOOL

Status of counter (CV ≥ PV)

CV

Data type: INT

Counter value

10.4.3 CTU_DINT up counter

The method of operation is the same as for the CTU up counter (Page 530) except for the following:

The CV and PV parameters are both DINT data types, which means that the maximum counter value possible is 2_147_483_647 (= 16#7FFF_FFFF).

Table 10-6 Parameters for CTU_DINT

Identifier	Parameter	Data type	Description
CU	Input	BOOL	Count up if value changes from FALSE to TRUE (rising edge)
R	Input	BOOL	Reset the counter to 0
PV	Input	DINT	Comparison value
Q	Output	BOOL	Status of counter (CV ≥ PV)
CV	Output	DINT	Counter value

10.4.4 CTU_UDINT up counter

The method of operation is the same as for the CTU up counter (Page 530) except for the following:

The CV and PV parameters are both UDINT data types, which means that the maximum counter value possible is 4_294_967_295 (=16#FFFF_FFFF).

Table 10-7 Parameters for CTU_UDINT

Identifier	Parameter	Data type	Description
CU	Input	BOOL	Count up if value changes from FALSE to TRUE (rising edge)
R	Input	BOOL	Reset the counter to 0
PV	Input	UDINT	Comparison value
Q	Output	BOOL	Status of counter (CV ≥ PV)
CV	Output	UDINT	Counter value

10.4.5 CTD down counter

The CTD counter allows you to perform downward counting operations.

- If the LD input = TRUE when the FB is called, then the CV output is reset to start value PV.
- If the CD input changes from FALSE to TRUE (from 0 to 1) when the FB is called (= positive edge), then the CV output is decremented by 1.
- Output Q specifies whether CV is less than or equal to 0.

The CV and PV parameters are both INT data types, which means that the minimum counter value possible is –32_768 (= 16#8000).

Table 10-8 Call parameters for CTD

Identifier	Parameter	Data type	Description
CD	Input	BOOL	Count down if value changes from FALSE to TRUE (rising edge)
LD	Input	BOOL	Reset the counter to start value
PV	Input	INT	Start value of counter
Q	Output	BOOL	Status of counter (CV ≤ 0)
CV	Output	INT	Counter value

10.4.6 CTD_DINT down counter

The method of operation is the same as for the CTD down counter (Page 532) except for the following:

The CV and PV parameters are both DINT data types which means that the minimum counter value possible is –2_147_483_648 (= 16#8000_0000).

Table 10-9 Call parameters for CTD_DINT

Identifier	Parameter	Data type	Description
CD	Input	BOOL	Count down if value changes from FALSE to TRUE (rising edge)
LD	Input	BOOL	Reset the counter to start value
PV	Input	DINT	Start value of counter
Q	Output	BOOL	Status of counter (CV ≤ 0)
CV	Output	DINT	Counter value

10.4.7 CTD_UDINT down counter

The method of operation is the same as for the CTD down counter (Page 532) except for the following:

The CV and PV parameters are both UDINT data types, which means that the minimum counter value possible is 0.

Table 10-10 Call parameters for CTD_UDINT

Identifier	Parameter	Data type	Description
CD	Input	BOOL	Count down if value changes from FALSE to TRUE (rising edge)
LD	Input	BOOL	Reset the counter to start value
PV	Input	UDINT	Start value of counter
Q	Output	BOOL	Status of counter (CV = 0)
CV	Output	UDINT	Counter value

10.4.8 CTUD up/down counter

The CTUD counter allows you to perform both upward and downward counting operations.

- Reset the CV count variable:
  - If the input is R = TRUE when the FB is called up, then the CV output is reset to 0.
  - If the LD input = TRUE when the FB is called, then the CV output is reset to start value PV.
- Count:
  - If the CU input changes from FALSE to TRUE (0 to 1) when the FB is called (rising edge), then the CV output is incremented by 1.
  - If the CD input changes from FALSE to TRUE (from 0 to 1) when the FB is called (= rising edge), then the CV output is decremented by 1.
- Counter status QU or QD:
  - Output Q specifies whether CV is greater than or equal to comparison value PV.
  - Output QD specifies whether CV is less than or equal to 0.

Table 10-11 Parameters for CTUD

Identifier	Parameter	Data type	Description
CU	Input	BOOL	Count up if value changes from FALSE to TRUE (rising edge)
CD	Input	BOOL	Count down if value changes from FALSE to TRUE (rising edge)
R	Input	BOOL	Reset the counter to 0 (up counter)
LD	Input	BOOL	Reset the counter to PV start value (down counter)
PV	Input	INT	Comparison value (for up counter) Start value (for down counter)
QU	Output	BOOL	Status as up counter (CV ≥ PV)
QD	Output	BOOL	Status as down counter (CV ≤ 0)
CV	Output	INT	Counter value

10.4.9 CTUD_DINT up/down counter

The method of operation is the same as for the CTUD up/down counter (Page 534) except for the following:

The CV and PV parameters are both DINT data types.

Table 10-12 Parameters for CTU_DINT

Identifier	Parameter	Data type	Description
CU	Input	BOOL	Count up if value changes from FALSE to TRUE (rising edge)
CD	Input	BOOL	Count down if value changes from FALSE to TRUE (rising edge)

Identifier	Parameter	Data type	Description
R	Input	BOOL	Reset the counter to 0 (down counter)
LD	Input	BOOL	Reset the counter to PV start value (down counter)
PV	Input	DINT	Comparison value (for up counter) Start value (for down counter)
QU	Output	BOOL	Status as up counter (CV ≥ PV)
QD	Output	BOOL	Status as down counter (CV ≤ 0)
CV	Output	DINT	Counter value

10.4.10 CTUD_UDINT up/down counter

The method of operation is the same as for the CTUD up/down counter (Page 534) except for the following:

The CV and PV parameters are both UDINT data types.

Table 10-13 Parameters for CTU_UDINT

Identifier	Parameter	Data type	Description
CU	Input	BOOL	Count up if value changes from FALSE to TRUE (rising edge)
CD	Input	BOOL	Count down if value changes from FALSE to TRUE (rising edge)
R	Input	BOOL	Reset the counter to 0 (up counter)
LD	Input	BOOL	Reset the counter to PV start value (down counter)
PV	Input	UDINT	Comparison value (for up counter) Start value (for down counter)
QU	Output	BOOL	Status as up counter (CV ≥ PV)
QD	Output	BOOL	Status as down counter (CV = 0)
CV	Output	UDINT	Counter value

10.5 Timers

Description

Timers are elements in your program used to execute and monitor time-driven processes. ST provides a series of system function blocks that you can access with ST. You can use timers in your program to do the following:

- Set waiting times
- Enable monitoring times
- Generate pulses
- Measure times

TP pulse

With a signal state change from 0 to 1 at the IN input, time ET is started. Output Q remains at 1 until elapsed time ET is equal to programmed time value PT. As long as time ET is running, the IN input has no effect.

The following figure illustrates the mode of operation of the TP pulse timer.

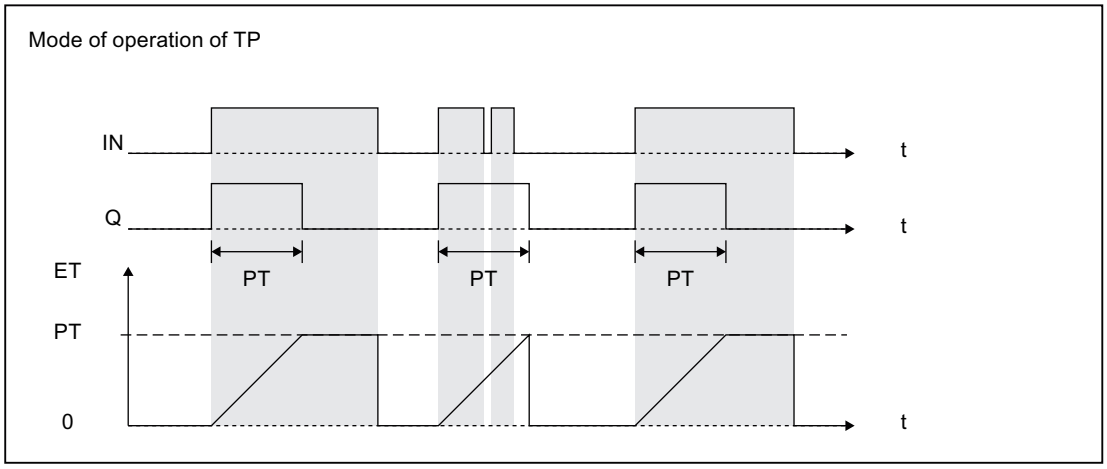


Figure 10-5 Mode of operation of the TP pulse

The following table shows the call parameters.

Table 10-14 Call parameters for TP

Identifier	Parameter	Data type	Description
IN	Input	BOOL	Start input
PT	Input	TIME	Duration of pulse
Q	Output	BOOL	Status of time
ET	Output	TIME	Elapsed time

TON ON delay

With the signal state change from 0 to 1 at the IN input, time ET is started. The output signal Q only changes from 0 to 1 if the time ET = PT has elapsed and the input signal IN still has a value of 1, i.e. the output Q is switched on with a delay. Input signals of shorter durations than that of programmed time PT do not appear at the output.

The following figure illustrates the mode of operation of the TON ON delay timer.

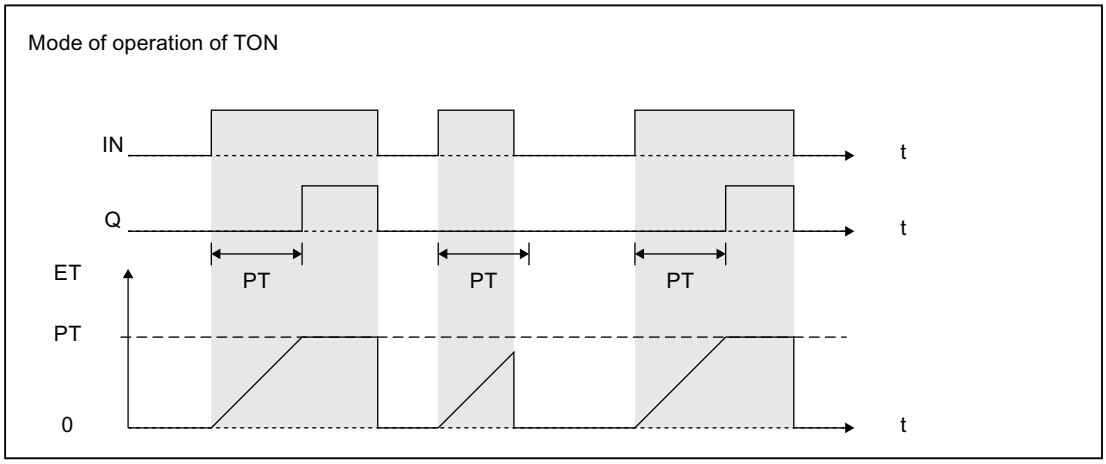


Figure 10-6 Mode of operation of the TON ON delay

The following table shows the call parameters.

Table 10-15 Call parameters for TON

Identifier	Parameter	Data type	Description
IN	Input	BOOL	Start input
PT	Input	TIME	Duration for which the rising edge at input IN is delayed
Q	Output	BOOL	Status of time
ET	Output	TIME	Elapsed time

Example

```
VAR
    Mytimeout : TON;
END_VAR

Mytimeout (PT := T#2s, IN := TRUE);
IF (mytimeout.Q) THEN
    ; //Time has expired
END_IF;
```

TOF OFF delay

With a signal state change from 0 to 1 at start input IN, state 1 appears at output Q. If the state at the start input changes from 1 to 0, then time ET is started. If a change occurs at input IN from 0 to 1 before time ET has elapsed, then the timer operation is reset. A start is initiated again when the state at input IN changes from 1 to 0. Only after the duration ET = PT has elapsed does output Q adopt a signal state of 0. This means that the output is switched off with a delay.

The following figure illustrates the mode of operation of the TOF OFF delay timer.

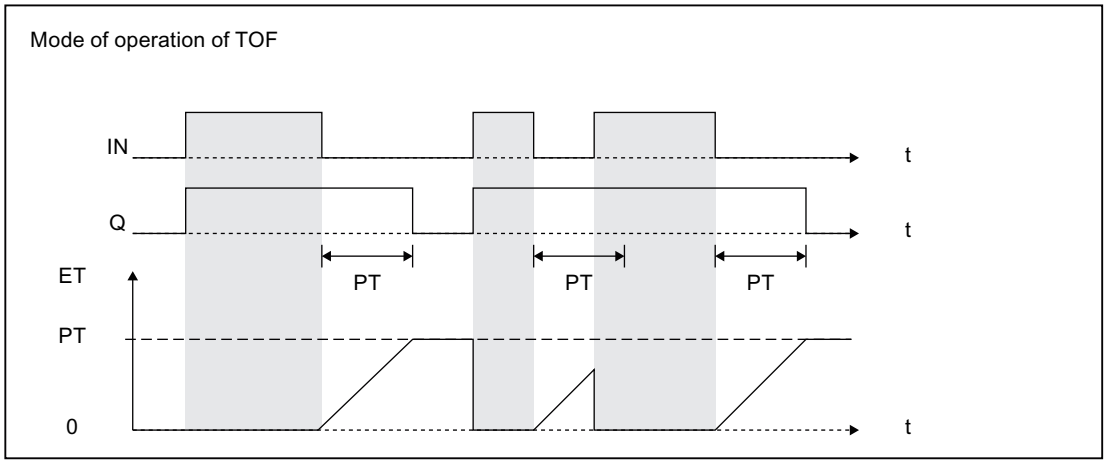


Figure 10-7 Mode of operation of the TOF OFF delay timer

The following table shows the call parameters.

Table 10-16 Call parameters for TOF

Identifier	Parameter	Data type	Description
IN	Input	BOOL	Start input
PT	Input	TIME	Period for which the falling edge at input IN is delayed
Q	Output	BOOL	Status of time
ET	Output	TIME	Elapsed time

Real-time clock RTC

The rising edge on SET sets the real-time clock to the value of PDT; PDT is copied to CDT. If READ is set to TRUE, then the current system time is read and is available at output CDT.

Table 10-17 Call parameters for the RTC

Identifier	Parameter	Data type	Description
SET	Input	BOOL	Set time, default FALSE
READ	Input	BOOL	Read time, default FALSE
PDT	Input	DT	Value to which the real-time clock is to be set, default DT#0001-01-01-0:0:0 If the value is less than the default value of the SIMOTION device real-time clock, the real-time clock will be set to its default value (e.g. with C2xx: DT#1994-01-01-00:00:00)
CDT	Output	DT	Current system time

The granularity for setting the real-time clock is 1 ms; the accuracy depends on the position control cycle clock.

10.6 Splitting bit-string data types

10.6.1 General information for splitting bit-string data types

The following functions enable a variable of a bit string data type to be split into multiple variables of a lower-level data type.

The inverse functions are implemented as functions, see Combining bit-string data types (Page 404).

10.6.2 _BYTE_TO_8BOOL function block

Description

This function splits a variable of data type BYTE into eight variables of data type BOOL.

User interface prototype

```
FUNCTION_BLOCK _BYTE_TO_8BOOL
    VAR_INPUT
        bytein : BYTE;
    END_VAR
    VAR_OUTPUT
        bit0,               // Least significant bit
        bit1, bit2, bit3, bit4, bit5, bit6,
        bit7 : BOOL;        // Most significant bit
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Input parameter

bytein

Data type: BYTE

Variable of data type BYTE to be split into eight variables of data type BOOL.

Output parameter

bit0
...
bit7

Data type: BOOL

Eight variables with the individual bits of the input parameter

bit0: Least significant bit
...
bit7: Most significant bit

10.6.3 _WORD_TO_2BYTE function block

Description

This function splits a variable of data type WORD into two variables of data type BYTE.

User interface prototype

```
FUNCTION_BLOCK _WORD_TO_2BYTE
    VAR_INPUT
        wordin : WORD;
    END_VAR
    VAR_OUTPUT
        byte0,           // Less significant byte
        byte1 : BYTE;    // More significant byte
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Input parameter

wordin

Data type: WORD

Variable of data type WORD to be split into two variables of data type BYTE.

Output parameter

byte0
byte1

Data type: BYTE

Two variables with the individual bytes of the input parameter

byte0: Less significant byte
byte1: More significant byte

10.6.4 _DWORD_TO_2WORD function block

Description

This function splits a variable of data type DWORD into two variables of data type WORD.

User interface prototype

```
FUNCTION_BLOCK _DWORD_TO_2WORD
    VAR_INPUT
        dwordin : DWORD;
    END_VAR
    VAR_OUTPUT
        word0,           // Less significant word
        word1 : WORD;    // More significant word
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Input parameter

dwordin

Data type: DWORD

Variable of data type DWORD to be split into two variables of data type WORD.

Output parameter

word0
word1

Data type: WORD

Two variables with the individual words of the input parameter

word0: Less significant word
word1: More significant word

10.6.5 _DWORD_TO_4BYTE function block

Description

This function splits a variable of data type DWORD into four variables of data type BYTE.

User interface prototype

```
FUNCTION_BLOCK _DWORD_TO_4BYTE
    VAR_INPUT
        dwordin : DWORD;
    END_VAR
    VAR_OUTPUT
        byte0,           //  Least significant byte
        byte1, byte2,
        byte3 : BYTE;    // Most significant byte
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Input parameter

in

Data type: DWORD

Variable of data type DWORD to be split into four variables of data type BYTE.

Output parameter

byte0
byte1
byte2
byte3

Data type: BYTE

Four variables with the individual bytes of the input parameter

byte0: Least significant byte

...

byte3: Most significant byte

10.7 Emulation of SIMATIC S7 commands

10.7.1 General

The following function blocks are interface-compatible with the commands for the SIMATIC S7 counters and timers; see the reference manual SIMATIC Statement List (STL) for S7-300/400.

Note

You should generally use the function blocks in accordance with IEC 61131-3 (**counters** and **timers**).

See also

General information on counters (Page 530)
Timers (Page 535)

10.7.2 _S7_COUNTER function block

User interface prototype

```
FUNCTION_BLOCK _S7_COUNTER
    VAR_INPUT
        CU: BOOL;
        CD: BOOL;
        PV : INT;
        S: BOOL;
        R : BOOL;
    END_VAR
    VAR_OUTPUT
        Q : BOOL;
        CV : INT;
    END_VAR;
    // ... (Code)
END_FUNCTION_BLOCK
```

Input parameters

CU
Data type: BOOL
Count up (with edge)

CD
Data type: BOOL
Count down (with edge)

PV
Data type: INT
Preset value

S
Data type: BOOL
Set preset value (with edge)

R
Data type: BOOL
Reset counter (static)

Output parameters

Q
Data type: BOOL
Counter status

CV
Data type: INT
Counter value

10.7.3 _S7_TIMER function block

This function block is used to emulate SIMATIC S7 time functions.

User interface prototype

```
FUNCTION_BLOCK _S7_TIMER
    VAR_INPUT
        T_TYPE : WORD;
        FR     : BOOL;
        S      : BOOL;
        R      : BOOL;
        TV     : TIME;
        TV_BCD : WORD;
    END_VAR
    VAR_OUTPUT
        Q      : BOOL;
        BI     : TIME;
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Input parameters

T_TYPE
Data type: WORD
S7 timer function to be executed
TYPE_S7T_SI_ = 16#0001 SI SP
TYPE_S7T_SV_ = 16#0002 SV SEE
TYPE_S7T_SE_ = 16#0004 SE SD
TYPE_S7T_SS_ = 16#0008 SS SS
TYPE_S7T_SA_ = 16#0010 SA SF
TYPE_S7T_BCD_IN = 16#0020 SA SF
TYPE_S7T_SI_ = 16#0040 R FR
TYPE_S7T_SI_ = 16#0080 L LC (U; UN; X ...)

FR
Data type: BOOL

Release

S

Data type: BOOL

Set preset value

R

Data type: BOOL

Reset timer

TV

Data type: TIME

Preset value (as data type TIME)

TV_BCD

Data type: WORD

Preset value in S7 coding

Output parameters

Q

Data type: BOOL

Timer status

BI

Data type: TIME

Current time

SIMOTION memory concept (in the target device) 11

11.1 Overview of the memory in the target device

Memory types

SIMOTION has a staggered memory management concept. A distinction is made in the target device between the DRAM (RAM disk and RAM) and the SRAM/NVRAM. The data (TOs, units and TPs) in the DRAM are lost when the device is switched off whereas the SRAM/NVRAM can retentively store smaller data quantities. A removable data medium (CF card or memory card) is also available as ROM.

RAM disk

The RAM disk is a virtual memory that can be controlled in the same way as a hard disk, but can be accessed much faster. A fixed memory area is reserved for the RAM disk in the DRAM of the target device. After the download, the RAM disk contains the hardware and device configuration, technology packages (TP), configuration data of the technology objects, and the program units. With **Copy RAM to ROM**, the contents of the RAM disk are written to the ROM (on the card). During an upload, the data is loaded from the RAM disk to the programming device. You can display the utilization of the RAM disk in the device diagnostics under System utilization (ONLINE only).

RAM

User data (user RAM) and system data (system RAM) are stored in the RAM. The user RAM contains the TO Current data memory, the TO Next memory of the technology objects (for more information, see Changing system variables and configuration data online (Page 550)) and is also used to store technology packages, units (variables and code). During power-up or download, the data is loaded from the RAM disk to the RAM.

The user RAM containing the TPs and units can be divided into a data memory (e.g. variables) and a code memory (compiled user programs) for clarity. Retain variables are saved retentively (powerfail-proof) in the SRAM/NVRAM.

The kernel, kernel data and further settings such as communication settings, cycle clock settings and the contents of the diagnostic buffer are stored in the system RAM.

You can display the utilization of the RAM in the device diagnostics under System utilization (ONLINE only).

ROM (CF card or memory card)

Data is saved retentively on the memory card (ROM). To do this, you must execute the command **Copy RAM to ROM**. The data saved last with Copy RAM to ROM is loaded from the ROM to the RAM disk during power-up and from there to the RAM. You can display the utilization of the CF/memory card in the device diagnostics under System utilization (ONLINE only).

SRAM/NVRAM (retentive data - non-volatile data)

Data is saved retentively and in a powerfail-proof manner in the SRAM/NVRAM. The data is loaded from the SRAM/NVRAM to the RAM during power-up. You can display the allocation of the SRAM/NVRAM in the device diagnostics under System utilization (retentive data, ONLINE only).

Non-volatile data available in SIMOTION devices:

Non-volatile data	Contents
Kernel data	Last operating mode
	IP parameters (IP address, subnet mask, router address)
	DP parameters (PROFIBUS DP address, baud rate)
	Diagnostics buffer
Retain variables	Variables in the interface or implementation section of a unit declared with VAR_GLOBAL RETAIN
	Local variables in programs, function blocks or classes declared with VAR RETAIN. As of SIMOTION Kernel version V4.5
	Global device variables set with the "RETAIN" attribute
Retain TO	Absolute encoder offset
DCC blocks	SAV blocks and user-defined blocks with retain behavior

The user program can use the "_savePersistentMemoryData" system function to save the contents of non-volatile data to the memory card. This prevents the retain variables and the absolute encoder position from being lost if, for example, a component is replaced (for more details, see the *System Function/Variables* List Manual).

The following graphic shows the general structure of the memory in the target device

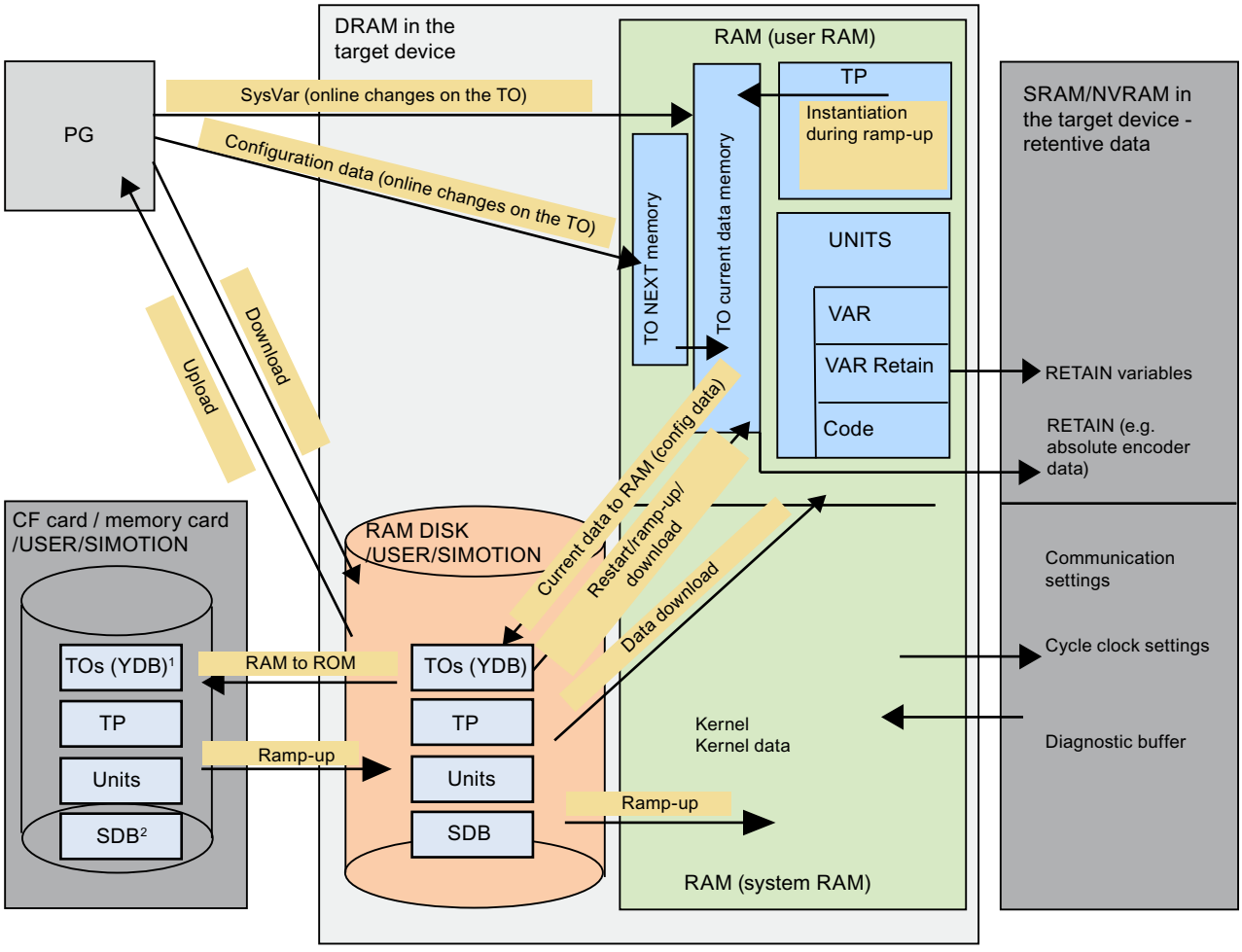


Figure 11-1 Memory concept, general representation

[1] YDBs contain the configuration data of technology objects, such as system variables, configuration data or alarms.

[2] SDBs are system data blocks

See also

Overview of the data download (Page 555)

Copy RAM to ROM (Page 589)

Download in RUN (Page 567)

11.2 Memory access

Power-up of the target device

The data of the TOs, technology packages, units and further data is transferred from the ROM via the RAM disk to the USER RAM during a power-up. The TO data is loaded to the TO current data memory, the technology packages (TP) and units to the USER-RAM. The individual TOs are instantiated and assigned the appropriate values. Communication settings, cycle clock settings, and the content of the diagnostic buffer (if available) are transferred from the SRAM/NVRAM to the system RAM, along with the retain data.

Note

IP and DP parameters in non-volatile data

If there is configuration data on the memory card, the IP and DP parameters are loaded from the memory card during ramping up and used by the SIMOTION device. The addresses defined in the configuration data are used to go online. During ramping up, the IP and DP parameters on the memory card are also written to the non-volatile data. If the SIMOTION device is then powered up with a CF card with no configuration, the IP and DP parameters are retained in the non-volatile data and are used by the device. This means the SIMOTION device can continue to go online if a configuration has been loaded with SIMOTION SCOUT at least once or if the SIMOTION device is powered up with a memory card containing a configuration.

Power On and overall reset

Data (configuration data, default values of system variables) is loaded from ROM (memory card) into the RAM disk and then into the TO current data memory. During the memory reset operation, the RAM and the retentive data in the SRAM/NVRAM, except for the communication configuration (baud rates, network addresses, etc.), are deleted prior to the power-up. In addition, the unit data is deleted during the memory reset and are then reloaded from the ROM during power-up.

Restarting a technology object

Data of the TO (configuration data, default values of system variables, cams) is loaded from RAM disk into the TO current data memory.

STOP - RUN transition

There is no access to any of the various memories during the transition from STOP to RUN.

Download to the target device

During a download from the programming device to the target device, the data is first written to the RAM disk and then from there to the RAM, with optional data initialization. See also Overview of the data download (Page 555) and Separate initialization of source and TO data during a download (Page 567).

Additional data (e.g. sources) can be included in the download to the target device.

This data is required for

- Online object comparison (e.g. additional properties)
- Various detailed comparisons (e.g. ST source file comparison)
- "Load to PG" function (complete upload to offline project)
- Synchronization with online objects

In order to be able to load sources or additional data of a project to the PG, this must be specified in the project under **Options > Settings > Download > Store additional data on the target device**.

Changing system variables and configuration data online

If you change the values of system variables, the system variable values are effective immediately in the TO current data memory. New configuration data values are initially stored in the Next memory. Immediately effective configuration data is transferred automatically to the TO current data memory. Configuration data that only becomes active following a RESTART on the technology object (set the *restartactivation* system variable to ACTIVATE_RESTART) is only written to the TO current data memory after the RESTART.

To save the configuration data changed online to the offline project, you must first transfer the content of the TO current data memory to the RAM disk with the menu command **Target system > Copy current data to RAM**.

The configuration in SCOUT is then no longer consistent with the configuration in the target device as the data of the RAM disk is used for the consistency check. Read the data from the RAM disk with the menu command **Target system > Load > Load to PG** (only for the configuration data) to re-establish a consistent system state. To ensure that the configuration is saved on the CompactFlash card or memory card with protection against power failure, execute the menu command **Target system > Copy RAM to ROM**. This menu command can also be used to implicitly execute the **Copy current data to ROM** and **Load to PG** functions.

As of V4.2, configuration data is completed during ramp-up using parameter values of the SINAMICS drive (adaptation). For more detailed information, see "Adaptation" under Symbolic assignment - introduction (Page 87). When "Copy current data to RAM" or "Copy RAM to ROM" is executed, adapted values are detected and "Load to PG" is automatically preselected as well.

Note

Copy current to RAM does not transfer the values of the system variables to the RAM memory. This means that "Save to memory card (Copy RAM to ROM)" or "Save in the engineering project (Load to PG)" is not possible.

So that values of system variables can also be saved in the engineering project and on the memory card, the default value of system variables must be changed OFFLINE and then loaded to the target device per download and saved.

For information on how TOs, variables and programs behave during a download in RUN, see Download in RUN (Page 567)

Backing up non-volatile SIMOTION data

You have the following options for backing up non-volatile SIMOTION data on the memory card:

- In the user program:
  The "_savePersistentMemoryData" system function allows the user program to back up the content of non-volatile SIMOTION data on the memory card. This ensures that the retain variables and the absolute encoder position are backed up in the event that a spare part is used.
- Using the switch/pushbutton (service selector switch or DIAG pushbutton the SIMOTION D) or IT DIAG.

Backing up non-volatile SINAMICS data

As of SINAMICS FW version V4.5, the backup of the non-volatile SINAMICS data (NVRAM data) is carried out by setting the CU parameter p7775 to the value 1. The data is stored on the CF card under /USER/SINAMICS/DATA/NVRAM/PMEMORY.ACX.

- For SIMOTION D4xx-2 with SINAMICS Integrated and CX32-2 controller extension
- For SINAMICS drives (e.g. S120) with CU310-2 or CU320-2 control units

Restoring non-volatile SIMOTION data

SIMOTION data backed up on a CompactFlash card with _savePersistentMemoryData is restored in the following cases:

1. After an overall reset
2. By selecting a switch position on SIMOTION D
3. SRAM content lost due to a discharged battery. In this case, the entire contents of the SRAM are restored from the file.

If, after a module has been replaced, retain data is available on the module, it is accepted and remains valid as long as the TO name or unit name is consistent. Otherwise, this data is invalid.

This means that once a module has been replaced, if the project is not consistent with the saved retain data, you should proceed as described in 1 or 2.

The backed-up data in the USER/SIMOTION/PMEMORY.XML file is copied back to the SRAM/NVRAM when the controller is being ramped up.

System variable device.persistentDataPowerMonitoring.persistentDataState indicates the state of the non-volatile SIMOTION data after ramp-up. The FROM_RAM state indicates that the non-volatile SIMOTION data has not been lost and has been downloaded from the SRAM/NVRAM.

Table 11-1 State of non-volatile data after ramp-up (persistentDataState system variable)

State	Meaning
FROM_RAM (1)	Non-volatile SIMOTION data in the SIMOTION device is used
FROM_FILE (2)	Non-volatile SIMOTION data restored from the backup file

State	Meaning
FROM_BACKUP (3)	Non-volatile SIMOTION data is restored from the backup copy of the backup file
INVALID (4)	Data in the non-volatile SIMOTION data and in the backup file / backup copy of backup file is invalid or not available/deleted. The SIMOTION device has copied the default settings to the non-volatile SIMOTION data and used this data to power up.

Restoring non-volatile SINAMICS data

The restoration of the SINAMICS-non-volatile data

- Is performed automatically in the event of a module replacement:
  A module replacement is detected on the basis of the serial number.
- Can be performed manually:
  Restore can be initiated manually by setting the CU parameter p7775 to 2.

Downloading data to the target device 12

12.1 Overview of the data download

Description

You have to download the project data that you have created with the SCOUT to the target system. The target system can contain several CPUs (SIMOTION controllers or drives). The project data contains the programs (ST, MCC, LAD/FBD and DCC) that you have created and compiled, the hardware configuration and the technology objects (with their technology packages) that you have created and parameterized.

You must reload a project to the target system, if you have:

- Created or changed programs
- Changed definitions of global variables or symbolic I/O variables
- Made changes to the execution system
- Changed technology object configurations

Download requirements

The following conditions must be met before the project can be downloaded to the target system:

- All cables must be plugged in and the interfaces configured
- All program sources are compiled (Project > Save and compile changes) or the program sources will be compiled automatically before the download.
- The project has been checked for consistency (Project > Check consistency)
- The system status is ONLINE (menu: Project > Connect to target system)

Note

The processing work increases with the size of the project. Plan the required components first in order to avoid any download errors. Note the technical specifications for the relevant device, technology object, communications interface, etc.

If an error occurs, transfer of data to the target system is aborted. The **Target system output** tab in which the transfer sequence and any errors are logged indicates the possible cause of the error.

Note

If a power failure occurs on the SIMOTION device during the download (RAM2ROM), a message will appear in the diagnostics buffer: Starting lockout set, after you have switched to RUN mode. Go online again and carry out the download once more.

In earlier versions, this can lead to various faults that cannot be corrected by the system. Unexplainable error messages may appear.
If this happens:
Go online again and carry out the download once more.

Download scope

You can either download the entire project or perform a download for a specific device.

- Downloading a project to the target system (all target devices) (Page 559)
- Downloading CPU/drive unit to target device (Page 562)
- Downloading a program subset and single units (sources) to the target device (Page 564)

Operating state

You can perform a download in the STOP mode, and also in the RUN mode.

Additional download options in OFFLINE mode

Additional functions in OFFLINE mode are also available for transferring the project data in SCOUT to the memory card:

- Download direct to memory card or hard disk, see Download direct to memory card or hard disk (Page 583)
- Device Update Tool, see Downloading using the device update tool (Page 585).

12.2 Save and compile

The project must be saved and compiled first before the download. SIMOTION SCOUT distinguishes three commands in the main project menu that act differently when saving and compiling:

- Save; the project is saved to the hard disk
- Save and compile changes
- Save and recompile all

In addition, there is the following command in the context menu of a source, a device and a library:

- Accept and compile (a single source)

Save

The project is saved to the hard disk. The changes are accepted into the project. No other processes, such as compilation or consistency checking, are initiated for the project.

Save and compile changes

On this command, the whole project is searched for changes. If a source is found which has been changed or has no compilation results, this and any linked sources are compiled and saved (e.g. during an FB call). Therefore only the changes are compiled. Use this command for day-to-day operations within a SCOUT version.

Save and recompile all

All sources of the entire project are recompiled with this command. DCC libraries are also recompiled.

The **Save and recompile all** command is suitable if you are entirely sure that all the old data from older SCOUT versions has been removed and should be replaced with new compilation results. It consists of the following steps:

- Project-wide deletion of all compilation results
- Recompilation of all objects (except libraries, see also AUTOHOTSPOT)

Use this command if you specifically want to convert a project from an earlier SCOUT version to a later one. All the error correction and optimization features are accepted into the new SCOUT as part of the process, Thus, the compilation results in SCOUT and SIMOTION RT can become inconsistent. The project navigator and object comparison then display the objects as "inconsistent" in ONLINE mode. You need to load the whole project to the target system in order to debug it.

Accept and compile

The Accept and compile command compiles the changes of the individual source and accepts them in the project. The data, together with the project, is only saved to the hard disk if you select **Save**, **Save and compile changes** or **Save and recompile all**.

Comparison of compilation results in the project and in SIMOTION RT

Before SIMOTION SCOUT V4.0, compilation results were compared on the basis of their time stamp. As of V4.0, SCOUT uses a hash function to compare compilation results. A hash function compares larger amounts of data based on the hash code. Compilation results are only displayed as "consistent" if their hash code is the same in both the project and SIMOTION RT. The hash code is only deemed to be the "same" if code and compiler (note the SCOUT version) are the same.

Information about errors

Detailed information about transfer and compilation errors is output during the compilation of single sources (OFFLINE **Compile/Check Output** and ONLINE **Target System Output**). Double-clicking the error message in the detail window causes the cursor to jump to the error location in the source. Correct the errors, for example, by:

- Changing the configuration of the technology objects
- Changing the programs

Repeat the steps as often as required until no more errors are displayed.

Note

You have the option to output detailed error information even for project-wide compilation (with Save and (re)compile all). To do this, execute **Options > Settings > Compiler > Create detailed message log during compilation of the project/devices**.

12.3 Editing a project with a more recent SCOUT version

Opening a converted project

The project was opened and edited with a more recent SCOUT version, and saved in the current project format. This can cause inconsistencies in online mode for objects that were not edited. It suffices to compile the project to avoid this problem. Click **Yes** to compile the project immediately. If you click **No**, you have the possibility to resolve the inconsistencies subsequently with the **Save and compile changes** function.

12.4 Performing a consistency check

Checking the project for consistency and compiling the project

Before you download the project to the target system, the program sources must be compiled without error and the consistency ensured. In this process, I/O addresses or TO configurations are checked, for example.

1. Select the **Project > Save and compile changes** menu.
   SIMOTION SCOUT compiles all changed sources.
   Further information about saving and compiling the project can be found in Save and compile (Page 556).
2. Select the **Project > Consistency check** menu.
   SIMOTION SCOUT checks, for example, whether all technology objects have been configured and that the sources have been compiled without errors. Prior to a download, a consistency check can be performed automatically if the corresponding option is selected under **Options > Download > Check consistency before loading**.
   During the download, a consistency check is always performed in the target system. If download errors occur, their possible causes are displayed by entries in various output windows (ONLINE in the **Target System Output** window) of the detail view.

Note

SINAMICS components for the consistency check

SINAMICS components are considered only for a cross-project consistency check. For the consistency check on the device, e.g. on the SIMOTION D4x5-2, the SINAMICS component (SINAMICS Integrated or Controller Extensions) is not checked.

12.5 Downloading a project to the target system (all target devices)

During a project download, the entire project is loaded to all devices that are connected online (you can tell which devices these are from the green or red/green plug icons in the project navigator). This loads data to all devices selected in the **Target Device Selection** dialog. This includes all devices which are located below a SIMOTION device (e.g. SINAMICS Int, external PROFIBUS/PROFINET slaves). If you go offline with a specific device prior to running the download, data will not be downloaded to that device. If you change the selection in the **Target Device Selection** dialog after going online, this does not affect the download at first (device remains connected).

Note

You can only perform a project download for all devices connected ONLINE and their subordinate drive units, in the STOP operating state.

You cannot download to drives that cannot be configured in SCOUT, such as MASTERDRIVE or 611U.

Note

Project change (downloading a new project to the target system)

Delete the user data on the CF card before downloading the new project.

We also recommend that you reset the controller to the factory settings.

The procedure is described in the Commissioning Manuals.

Procedure

1. Click the **Download project to target system** button. The **Download to Target System** dialog box is displayed.

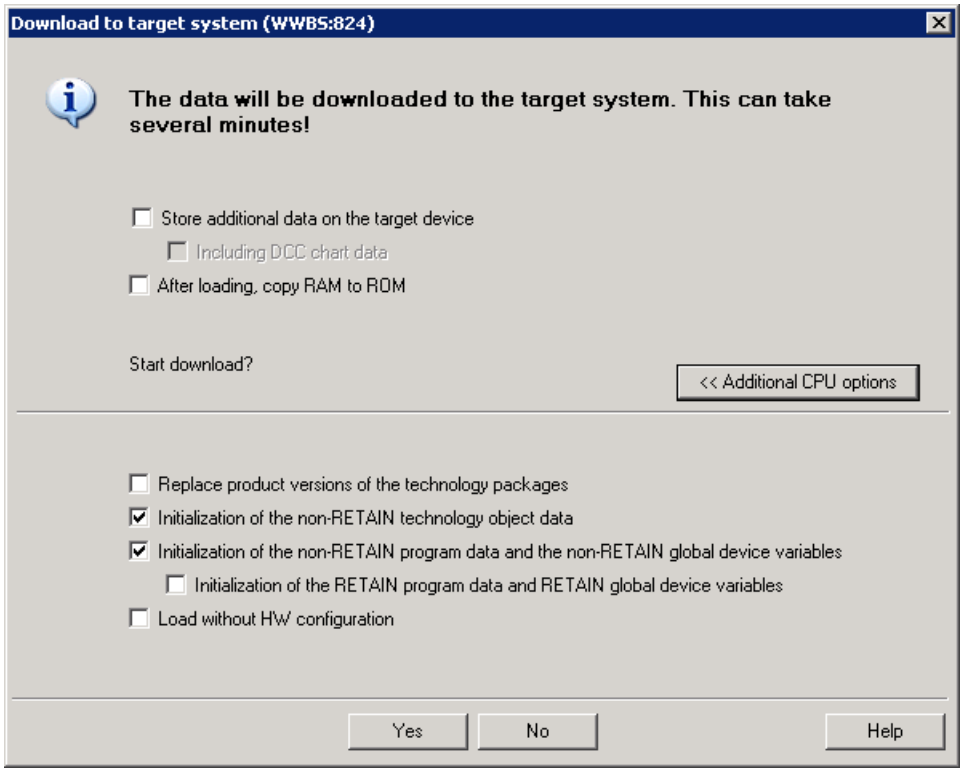


Figure 12-1 Download to target system

You can select the following options depending on the device:

- **Store additional data on the target device**; you can use this option to store supplementary data (e.g. program sources) on the target device.
  - **Including DCC chart data**
    With DCC, it is possible to store graphical chart data in addition to the supplementary data that enables a functional chart to be uploaded, meaning that the chart in question is in the full source text and can, therefore, be edited further.
  - For further information, see also Project download - settings in the SCOUT online help.
- **After loading, copy RAM to ROM**; copies RAM to ROM after the download for all target devices.
  After the download, the data is copied from RAM to ROM for all target devices. The loaded configuration is then stored retentively on the memory card.

Click **Additional CPU options** to display the following options:

- **Replace product versions of the technology packages**; this loads the technology package versions selected in the Select technology packages dialog to the target system. See also Downloading the selected product version of the technology packages (Page 566).
- **Initialization of the non-RETAIN technology object data** (see Separate initialization of source and TO data during a download (Page 567))
- **Initialization of the non-RETAIN program data and the non-RETAIN global device variables** - only effective in STOP mode
  - **Initialization of the RETAIN program data and RETAIN global device variables** (see Separate initialization of source and TO data during a download (Page 567))
- **Load without HW configuration** (see Download without HW Config information (Page 579))

The global settings are made at **Options > Settings > Download**.

Accepting or rejecting settings

- Click **Yes** to start the download or **No** to cancel the download.

The additional options are also active when they are not visible. The settings with which the last download was performed are then used. The only exception is the **Replace product versions of the technology packages** option which is only active for one action.

For drive units, you can only select **After loading, copy RAM to ROM** and **Supplementary data... incl. DCC chart data**.

Troubleshooting possible errors from older SCOUT versions:

If an error occurs when using projects from older SCOUT versions that can be reproduced in those versions, the error status can be resolved for the affected device and the project re-downloaded using the **Save and recompile all** command. If necessary, run **Delete user data on card** first.

Error messages:

- "IOM configuration inconsistent" (although there is no overlap of I/O variables).
- "UPP transfer of the source failed"

See also

Save and compile (Page 556)

12.6 Downloading CPU/drive unit to target device

Description

In addition to a project download, you can also selectively load data to each CPU / drive unit. The standard procedure is the download in STOP mode. You can also perform a download in RUN operating state. For further information, see Download in RUN (Page 567).

Procedure

1. First select the device to which you want to load the data in the project navigator (device must be ONLINE).
2. Click the **Download CPU / drive unit to target system**
   or
   right-click the device in the project navigator and perform **Target device > Download** in the context menu.
3. The **Download** dialog box is displayed. The contents depend on the device for which you want to download data.

Loading the CPU to the target device

A download can be performed for the entire CPU (without drive unit) but also separately for individual related download units.

- Download CPU without drive unit
- Download all technology objects of the CPU
- Download all programs of the CPU

You can select the following options:

- **Store additional data on the target device**; you can use this option to store supplementary data (e.g. program sources) on the target device.
  - **Including DCC chart data**
    With DCC, it is possible to store graphical chart data in addition to the supplementary data that enables a functional chart to be uploaded, meaning that the chart in question is in the full source text and can, therefore, be edited further.
  - For further information, see also Project download - settings in the SCOUT online help.
- **Copy RAM to ROM after download**
  After the download, the data is copied from RAM to ROM for the selected target device. The loaded configuration is then stored retentively on the memory card**.**
- **Perform download during RUN**; enables a download to be performed in RUN mode.
  Click **Additional CPU options** to display the following options:

- **Replace product versions of the technology packages**; this loads the technology packages selected in the Select Technology Packages (Page 566)dialog to the target device.
- **Initialization of the non-RETAIN technology object data** - only effective in STOP mode (see Separate initialization of source and TO data during a download (Page 567))
- **Initialization of the non-RETAIN program data and the non-RETAIN global device variables** - only effective in STOP mode
  - **Initialization of the RETAIN program data and RETAIN global device variables** (see Separate initialization of source and TO data during a download (Page 567))
- **Load without HW configuration** (see Download without HW Config information (Page 579))

You set the global default under **Options > Settings > Download** or **CPU download**.

The additional options are also active when they are not visible. The settings with which the last download was performed are then used. The only exception is the **Replace product versions of the technology packages** option which is only active for one action.

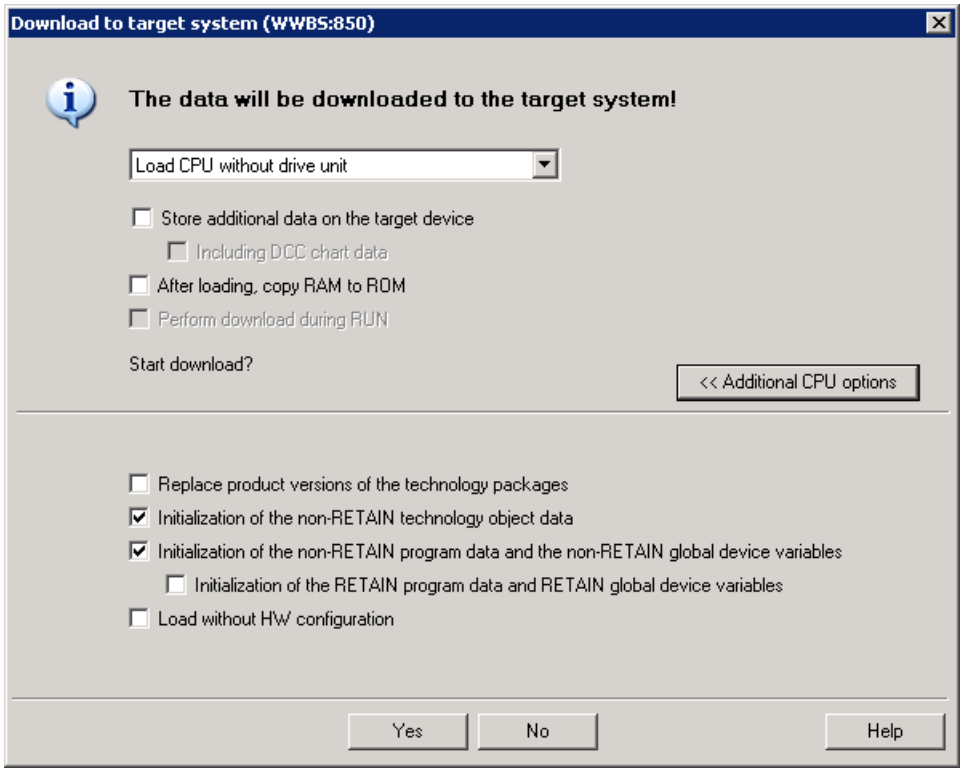


Figure 12-2 Download to CPU / drive unit

Accepting or rejecting settings

- Click **Yes** to start the download or **No** to cancel the download.

Downloading the drive to the target device

For drives, you can only download the drive data (parameterization, etc.) to the drive unit.

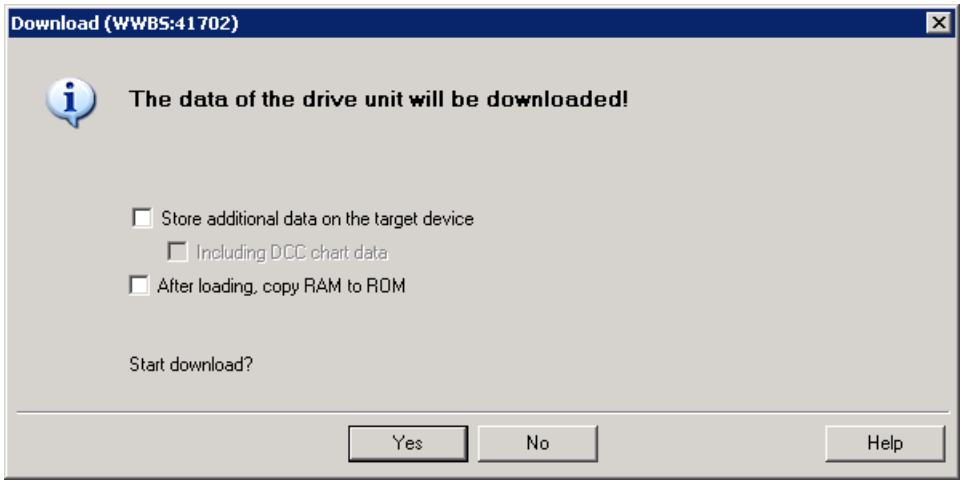


Figure 12-3 Download to drive unit

By default, the download is performed in STOP. After a successful download, the CPU can be restored to its previous status.

You can select the following options:

- **Store additional data on the target device**; you can use this option to store supplementary data (e.g. program sources) on the target device.
  - **Including DCC chart data**
    With DCC, it is possible to store graphical chart data in addition to the supplementary data that enables a functional chart to be uploaded, meaning that the chart in question is in the full source text and can, therefore, be edited further.
- **After loading, copy RAM to ROM**; copies RAM to ROM to the selected device once the download is complete.

See also

Download of changed technology objects (Page 580)

12.7 Downloading a program subset and single units (sources) to the target device

Download of selected units

As of SIMOTION V4.4, you can download a single of several units to the SIMOTION device in the RUN and STOP states.

You no longer have to download all units that have been displayed as inconsistent, but can download a specific program subset. This enables, for example, two users who are connected

online to target device at the same time with SCOUT to download separate program subsets independently.

To do this, right-click the unit in online mode and execute **Download...** in the context menu.

Note

The selective download is not possible for DCC charts.

When downloading a subset, the system ensures that the project as a whole is available again in a consistent state. For this purpose, it may be necessary to download additional units for the selection. This operation must be confirmed in a dialog box.

Note the following applications in this respect:

- You change the interface of a unit and this interface is used by a second unit. This unit is then also downloaded.
  Example: You change the signature of a function.
- You load only a subset of the changed units. After checking the dependency, independent units can be downloaded as a subset.
- You are referencing a library that has not been downloaded, in a unit that you want to download, or you change the interface of a library used in the unit. The library is downloaded with the unit.
- You delete a program (POU) of a unit that is called from the execution system. When downloading the unit, the execution system must also be downloaded.
- If programs have been downloaded to the controller that are not in the project, these programs are removed from the controller during a complete download. If you only load a subset, then programs are only deleted when there are no longer any dependencies.
  If the programs that have not been removed reference the programs that are to be downloaded (uses), then the behavior of these programs may be changed.
  Therefore, perform a detailed comparison to display the programs online and offline.

Selective download not possible

A selective download is not possible for the following reasons:

- If the relevant CPU has not been compiled error-free or the consistency check is not error-free.
- If an I/O variable has been changed in the address list.
- If a global device variable has been inserted, deleted, renamed or changed structurally (e.g. data type changed, length of the array changed); a change of the initial value does not prevent the download.
- If a TO has been inserted, deleted, renamed or changed structurally.
- If technology packages have to be loaded.
- If DCC is affected, see above.
- If tasks have been activated, deactivated for an execution system, or the cycle clocks have been changed.

12.8 Downloading the selected product version of the technology packages

Description

As of Version V4.2, you can select the product version of technology packages within a particular version (service packs and hotfixes). **Replace product versions of the technology packages** is used to download the selected technology package versions to the target system. This enables you to overwrite newer or older versions of the technology package in the target system. Further information can be found at Selecting or replacing technology packages in the SCOUT online help.

How to load a specific version of a technology package to the target system

1. Right-click a SIMOTION device in the project navigator and then click **Select Technology Packages**.
   The dialog opens.
2. Select the technology package you want to update.
3. Select the version under TP version and Version. Hotfix versions can also be selected here as of V4.2.
4. Click **OK** to close the dialog and accept these settings.
5. Go online with the device and click **Download project to target system** or **Download CPU to target system**.
   The dialog opens.
6. First click **Additional options** and select **Replace product versions of the technology packages**, then click **OK.**
7. The technology package is downloaded to the target system, even if a technology package already exists there.
   The SCOUT's detail view shows the loading process for the technology packages.

Boundary conditions for replacing technology packages

- If no technology package version is selected in the **Select Technology Packages** dialog, the latest technology package of the appropriate firmware version is downloaded to the target system.
- If you are performing an upload, the versions of the technology packages already present in the target system are uploaded too, entered on the device, and appear in the **Select Technology Packages** dialog.
- Technology package versions are exported and imported at the same time.
- If the version selected is not already on the PG/PC, an error message is output.

See also

Downloading a project to the target system (all target devices) (Page 559)

Downloading CPU/drive unit to target device (Page 562)

12.9 Separate initialization of source and TO data during a download

Description

As of V4.1.2, you can initialize non-RETAIN and RETAIN data of programs (units) and global device variables as well as non-RETAIN and technology objects separately via two settings. The data is initialized together at the first or complete download of the project. During servicing or if you only want to reload programs (units, sources) or technology objects, you can initialize the data separately.

- Initialization of the non-RETAIN technology object data; only non-RETAIN data can be initialized for TOs. This does not affect the program data (variable values).
- Initialization of the non-RETAIN program data (variables) and the non-RETAIN global device variables The RETAIN program data and RETAIN global device variables can also be initialized. This does not affect the technology object data. The global device variables are initialized together with the settings for the program data (variable values).

The initialization only takes effect in the STOP operating state.

To make the settings, see Downloading CPU/drive unit to target device (Page 562).

The "Time of variable initialization" section in the programming manuals contains general information on variable initialization.

12.10 Download in RUN

12.10.1 General information for downloading in RUN

Perform download in RUN

As for a download in STOP, all selected changes are always downloaded in RUN. Therefore, only manageable changes should be made for a download in RUN. You can load an entire target device or even just parts of the target device.

A number of issues presented below must be noted for a successful download in RUN

Allowing a download in RUN

Settings in SCOUT

1. In SCOUT, select **Enable download / copy RAM to ROM during RUN** under **Options > Settings > CPU download**. This activates the option **Perform download during RUN** in the **Download to target system** dialog.
2. Click the option **Perform download during RUN**.

12.10.2 Download of changed sources in RUN

12.10.2.1 Change to sources – general

If a source (unit) contains several programs, FBs or FCs (POUs), then the entire unit is always loaded.

A unit can generally be exchanged in RUN. At the time of transfer, neither code (of the programs, FBs or FCs) nor the data (variables) of this unit may be used in a task.

Units are loaded at the cycle control point.

At this point in time, all cyclic tasks are restarted. As no source is accessed at the time, no source is therefore blocked in cyclical tasks for exchange in RUN.

If Motion Tasks are also used and they have been started at the time of exchange, the system determines at the cycle control point which POUs are currently accessed and blocks the applicable units from exchange in RUN.

Several seconds are spent attempting to load the changed units simultaneously at the cycle control point. If that is not possible, the process is terminated.

Options for download in RUN:

- The number of cyclic tasks in which programs (POUs) can be loaded in RUN is limited to four.
- If the number of sources (units) and relevant tasks is too high for time or volume reasons, the sum of the changes cannot be processed (loaded). This problem is displayed in the output window and the complete set of changes is not transferred to the sequence. The old data remains effective.

12.10.2.2 Dependencies on other sources

SIMOTION applies the unit concept, so that you can access the global variables, data types, functions (FCs), function blocks (FBs), and programs of other source files.

You can connect to other program sources to use their exported components by using a USES instruction in ST sources or via the **Connections** tab on the declaration tables of MCC and LAD/FBD sources.

Changes to connected sources also affect download in RUN. Additional units may have to be loaded.

If units depend on another changed unit, these units must also be compiled before download. They may use data or POUs that have been changed. On download, these units must only be exchanged if data or POUs are used.

As of SIMOTION SCOUT V4.4, the dependencies between the units are taken into account. Only called (used) POUs require the unit to be replaced. There are therefore significantly fewer dependencies compared to older SCOUT versions. This behavior applies to all supported RT versions that are processed using SIMOTION SCOUT V4.4.

In projects that have been converted from older versions of SCOUT, the feature is only available after full re-compilation using SCOUT V4.4. Execute the function **Save and recompile all** on the **Project** menu for this purpose. Note that recompilation in this case creates an online inconsistency.

In the event of a simple change to the code of a POU and in the event of any changes to the data in the implementation part of a unit, only the modified unit has to be replaced. Please note that data changes in RUN are only possible if the data is initialized (see Changes to the data declaration (Page 571)).

Reduction of dependencies between the units

In the Engineering System, individual parameters of the units are checked for changes as of SIMOTION SCOUT V4.4.

The decision regarding whether changes in a used unit are relevant to your unit reduces the dependencies and therefore the necessary number of units to be replaced.

The following parameters are changes when a unit is used:

- Data declarations (if data from the unit is used)
- POU interfaces of used POUs (if POUs from the unit are used)
  - The following is checked for FUNCTION, FUNCTION_BLOCK and PROGRAM: Return value, VAR_INPUT and VAR_IN_OUT
  - For FUNCTION_BLOCK and PROGRAM the following are also checked: VAR and VAR_OUTPUT

Example 1

If the code of unit_2 is only processed in the BackgroundTask, the modified unit_2 can be replaced at time t4. However, as unit_1 has started a call for code (POU) from unit_2 that at time t4 has not yet returned, unit_2 cannot be replaced at this point in time.

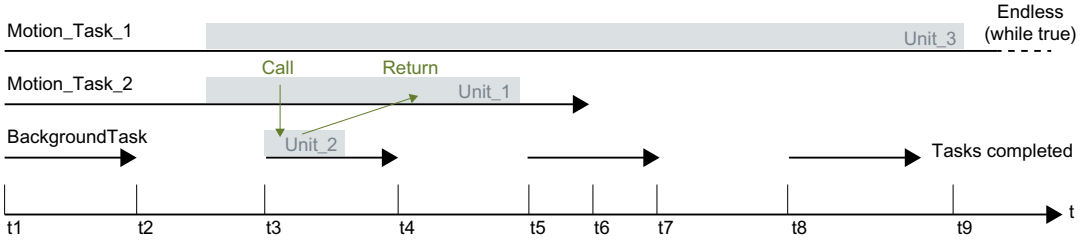


Figure 12-4 Dependencies on other sources – example 1

Unit_2 can be replaced at time t7 at the earliest.

A unit also cannot be replaced for as long as it is being processed by a MotionTask. In the example, unit_3 in the endless MotionTask can never be replaced.

The Motion_Task_1 can be stopped explicitly by the user to allow replacement to take place.

Example 2

In unit ST_2, data (data types and variables) and POUs (programs, functions, function blocks) used by ST_1 are indicated via the interface area:

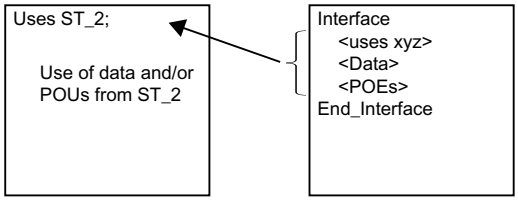


Figure 12-5 Dependencies on other sources – example 2

A change to the interface area of ST_2 has an impact on ST_1:

- ST_1 must also be re-compiled
- If ST_2 is to be replaced (loaded), ST_1 must also be replaced. Replacement depends on the scope of the change (see above) and the method for checking dependencies.

If, for example, a program is running in a cyclic task in ST_2 and a program is running in an endless MotionTask in ST_1, the unit ST_2 cannot be replaced in RUN. Unit ST_1 would also have to be replaced.

12.10.2.3 System unit of the execution system

Also note the system unit of the execution system in addition to the user units to be loaded This system unit primarily implements the execution system configuration and is therefore the connecting element between the programs and the tasks.

In the event of the following changes, the system unit must also be replaced:

- Changes to the program assignment and the size of the local data stacks in the execution system
- Changes to the data declaration of programs that are used in the execution system (PROGRAM container)
  (This point does not apply if the following compiler option is not selected for the unit that provides the program: "Create program instance data only once")
- Where a UserInterruptTask has been configured with condition on a global variable of a unit (from the interface area), if the interface (data structure of exported POUs) of this unit changes.

Where the system unit of the execution system is replaced, no task may be running simultaneously (long-running tasks and endless MotionTasks are problematic).

As of SIMOTION SCOUT V4.4, the Engineering System checks whether changes in the interface area of a user unit only related to the interfaces of the published programs or to the conditions of UserInterruptTasks. Only then is a change of the system unit of the execution system necessary, which must then be loaded and replaced. This procedure applies to all RT versions.

12.10.2.4 Changes to the data declaration

Changes to the following variables declarations are generally not permitted for a download in RUN, and the download in RUN is terminated. At the time of loading, the current values would otherwise have to be saved for all variables and, after the new variable structure had been created, the existing current values would have to be set as active again. This is not possible during the short period of loading.

If applicable measures are taken (permit initialization of the current variable values), download in RUN can, however, generally be performed anyway:

- Changes in VAR_GLOBAL blocks.
  Remedy: Permit initialization via compiler pragma BlockInit_OnChange or pragma lines in the declaration tables, see Recommended settings for download of changed sources in RUN (as of SIMOTION V4.1) (Page 573).
- Changes in VAR blocks by programs (program instance data) that are used in cyclic tasks. The program instance data of the cyclic task is retained over the run of the tasks and therefore cannot be changed.
  Remedy: Permit initialization via compiler pragma BlockInit_OnChange or pragma lines in the declaration tables and additionally set the compiler option "Only create program instance data once" at the unit that declares the program (PROGRAM). See Recommended settings for download of changed sources in RUN (as of SIMOTION V4.1) (Page 573).

The following variables declarations can always be changed:

- VAR_TEMP in programs and FBs, as well as all declaration blocks in FCs
- The program instance data of non-cyclic tasks (MotionTasks), as they are reinitialized on every start.
  (This only applies if the compiler option "Only create program instance data once" has not been activated for these programs (PROGRAM).)

12.10.3 Programmer notes for download of changed sources in RUN

Appropriate programming for download of changed sources in RUN

Use appropriate programming to enable a download of changed units in RUN mode.

- When structuring the units, ensure that data and POUs that are used both by cyclic and by sequential tasks are stored in separate units. This reduces the number of dependencies.
- Use the USES instruction (connection to other program sources to use their exported components) where possible in the implementation section and not in the interface section. This ensures there are fewer dependencies between the units, as USES instructions are not inherited in the connected source. Fewer units are affected, even at the compile stage. Changes therefore have an impact on fewer units. For more information, see "*USES statement in an importing unit*" in the *ST programming manual*.
- Configure the size of the local data stack in the task configuration for large sources (see Configure execution system (Page 244) and Setting size of local data stack (Page 597)): Take into account a reserve for download in RUN to avoid changes in the execution system. For example, additional storage may be required on the local data stack as the result of a change (e.g. for new local variables and function parameters).

In addition, for new variables:

- An additional variables declaration block may be used in the interface and implementation section of a unit with new declarations for TYPE and global variables. The variables declaration block must be introduced after the pre-existing declaration sections.
- Instead of using an additional variables declaration block, you can also create a new UNIT with the new data and then link it with USES (in the implementation section). This is also possible in MCC and LAD/FBD.

In addition, when using MotionTasks:

- Create units in accordance with how tasks and programs are assigned to one another. This minimizes dependencies and increases programming clarity.
  Negative example: One unit contains two programs. One of the two is assigned to the BackgroundTask and the other to a MotionTask.
  Download in RUN in this and other units is not possible while the MotionTask is active.
- If the program of the modified unit is assigned to a MotionTask: Provide options so that the MotionTask is not active when you want to replace the unit:
  - Avoid continuous loops in MotionTasks! For example, instead of a WHILE loop, use the *_restartTaskId* function.
  - Make provision for an operator function (such as single clock mode) so that you can reset one MotionTask, several MotionTasks, or all MotionTasks. MotionTasks can also be reset via SCOUT as of V4.1.2 (see Manage MotionTasks via SCOUT (as of V4.1.2) (Page 578)).
- Store the program parts in their own units, if the units run permanently or are called ad hoc. This is particularly advantageous in the case of MotionTasks. A sequential control system in MCC, for example, is a permanently running program part.
  These called program parts can be replaced as long as they are not being accessed.

12.10.4 Error messages if download is not possible

Description

Error messages are output during the download if a change is not possible in RUN, e.g. UPP change in RUN not possible (UPP = User Program Processing). UPP indicates that an error has occurred when a user program is loaded. The old program is retained in the controller, the change is rejected.

Any changes made to the source in the SCOUT project can be easily undone by reloading the current data on the controller to the Engineering System. This ensures the SCOUT project is consistent online once more.
Requirement: Supplementary data have previously been loaded at the same time.

See Loading data from the target device to the programming device (PG)/PC (Page 586).

12.10.5 Recommended settings for download of changed sources in RUN (as of SIMOTION V4.1)

12.10.5.1 Overview of settings

Download in RUN can be further improved by the following settings:

- Set the compiler switch "Create program instance data only once"
  The system unit of the execution system is then not affected by changed instance data of a program.
- Use the "BlockInit_OnChange := True;" compiler pragma or as of SCOUT V4.2, an appropriate pragma line in the declaration tables (MCC, LAD/FBD) to allow changed variables to be initialized for download in RUN.

12.10.5.2 Compiler option "Only create program instance data once"

Description

The program data instantiation and the storage of the program instance data are important in the context of download in RUN.

Compiler option "Only create program instance data once" is set (preferable for download in RUN)

- Instance data associated with programs compiled in this way is created just once, even if the program is used in multiple tasks. The instance data is then in the source or in the program code (for diagnostic purposes, this data is in the symbol browser for the unit and no longer under the execution system).
  If several tasks are assigned to one program, the same instance data is used as the basis for all of them.
  This setting has advantages for a download in RUN as the system unit of the execution system remains unchanged with changed instance data of a program.
- Local program variable data initialization is performed for sequential and cyclic tasks once when the source (unit) is downloaded and the CPU ramps up.
- A compiler pragma or the device properties can be adjusted so that the local variables are initialized on each STOP-RUN transition. See Initialization of data during a STOP - RUN transition (Page 576).
- Where instance data has been changed, a program within a cyclic or sequential task can even be downloaded in RUN while other tasks (cyclic or sequential) are active.
  Requirement: Allow initialization by compiler pragma BlockInit_OnChange or pragma lines in the declaration tables

NOTICE
Changed behavior for data initialization Changed behavior for the data initialization when "Only create program instance data once" has been selected (see above).

Information on which tasks can run sequentially or cyclically can be found under Execution levels / tasks (Page 199).

Compiler option "Only create program instance data once" is not activated (default)

- The instance data of all programs is in a central memory area (for diagnostic purposes, this data is in the symbol browser for the execution system). If several tasks are assigned to the same program, then separate program instance data will be created for each task.
  If instance data of a program has been changed, all non-cyclic tasks must be terminated (via central storage in the system unit of the execution system) for activation to be successful. This gives rise to restrictions for download in RUN.
- The data initialization of local program variables is performed:
  - For non-cyclic tasks, at start of task.
  - For cyclic tasks, at STOP-RUN transition.
- A program within a non-cyclic task can also be downloaded in RUN (as long as this task and other non-cyclic tasks are not active) even if the instance data is changed, as the program data is initialized at the start of the task.

Setting of the compiler option

- For the global default of the compiler settings for the program sources, activate **Only create program instance data once** under **Options > Settings > Compiler**.
- For the local setting, when inserting a program source, select **Only create program instance data once** under Compiler in the **Insert xxx** or **Properties** dialog of the source (xxx = ST, MCC, LAD, FBD unit).
  The local settings overwrite the global settings.
  The compiler switch must be placed in RUN before a download (i.e. the source must have been loaded previously with a download in STOP).

See also Influence of the compiler on variable initialization (Page 322).

12.10.5.3 Compiler pragma for re-initialization of program instance data for download in RUN

Description

Pragma lines in the declaration tables or the pragma "{BlockInit_OnChange := TRUE;}" in ST sources can initiate a re-initialization of the data with the default values specified in the source in the event of changes to the block structure during a download in RUN.

The pragma can be used in the following cases:

- Changes to TYPE and global variables (VAR_GLOBAL) in the interface and implementation section of a unit
  (alternatively an additional variable block can also be used)
- Changes to instance data of programs (VAR)
  (when the “Only create program instance data once” option is active)

Changes made in VAR_GLOBAL blocks in the INTERFACE section may affect other sources, which will then also need to be loaded. This applies to sources which import the amended source via USES (ST) or are linked with the changed source in the declaration table (MCC,

LAD/FBD) and access variables or POUs of this source. By contrast, only the changed source is affected during the download where changes are made in VAR_GLOBAL blocks in the IMPLEMENTATION section.

Where changes are made in VAR_GLOBAL blocks in the INTERFACE section, any MotionTasks which may be running prevent download in RUN despite the pragmas mentioned above (for details of how to avoid this, see Manage MotionTasks via SCOUT (as of V4.1.2) (Page 578)).

The pragma may only be placed at the beginning of a block.

Example of the syntax of BlockInit_OnChange:

```
VAR_GLOBAL
    {BlockInit_OnChange := TRUE;}
    Interface_Var_Global1 : INT;
    Interface_Var_Global2 : REAL;
END_VAR
```

NOTICE
Unexpected behavior after initializing data areas in RUN Initializing data areas in RUN can cause unexpected behavior, as any saved values from the previous task cycle are no longer available.

In LAD/FDB and MCC, initialization can be activated in the declaration tables by inserting pragma lines (context menu); see the relevant LAD/FDB and MCC Programming Manuals. The pragma can be set and deleted as required at any time (see also *Adding pragma lines in variable definition* in the MCC and LAD/FBD Programming Manuals).

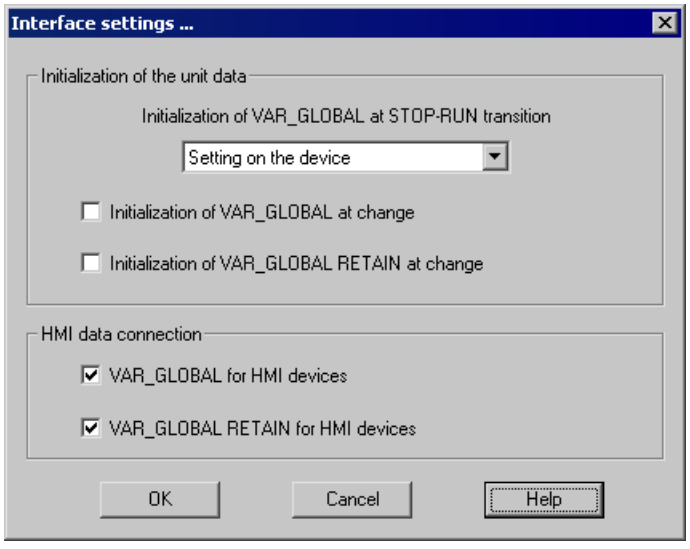


Figure 12-6 Pragma settings for unit variables (interface section)

Example

If it is not possible to perform a download in RUN because of a change to a VAR block within the program, the pragma can be subsequently set in the VAR block. Following compilation, a download in RUN will be possible, because the VAR block will have been reinitialized.

The pragma can be removed again prior to the next download. Where this is not the case, however, re-initialization in RUN is only performed in the event of a further change in the VAR block.

Simple changes to the pragma (add/delete) can be loaded in RUN at any time. You are not required to stop sequential tasks.

12.10.6 Initialization of data during a STOP - RUN transition

Description

If the "Only create program instance data once" compiler option has been activated, data initialization for program variables only occurs during a download or when the CPU is started.

You can set the configuration so that global unit variables (in the interface and implementation section) and program variables are also initialized during a STOP-RUN transition. The initialization is performed immediately before the start of the StartupTask. You therefore always have the ability to perform an initialization of the variables during a STOP-RUN transition:

- As of V4.2 via the Device properties dialog.
- As of V4.1.2 with the compiler pragma "BlockInit_OnDeviceRun" in the Units (ST) or pragma lines in the declaration tables of the sources or programs/charts (MCC and LAD/FBD).

Initialization via the Device properties dialog (as of V4.2)

The **Initialization of the non-RETAIN global variables (VAR_GLOBAL and global device variables) and program variables (VAR) at the transition from STOP to RUN** option in the Properties dialog of a device enables you to trigger initialization immediately before the StartupTask is started.

This global setting can be overwritten locally with a compiler pragma or a pragma line (see below).

Always / never initialize data in ST sources

In ST sources, the initialization can be influenced during a STOP-RUN transition using the compiler pragma "BlockInit_OnDeviceRun" at the start of the variable blocks.

If you use this pragma, the setting on the device is no longer valid for the applicable variable block.

If you set this pragma to "ALWAYS" in the VAR_GLOBAL block or in the VAR block of programs, the variables are initialized at each STOP - RUN transition. The setting is only valid for the variable block in which the pragma is used.

Set the pragma with the "DISABLE" setting to prevent initialization at the STOP - RUN transition in a targeted way.

The compiler issues a warning when the pragma is used at a position where it has no effect.

See also Section "Controlling compiler with attributes" in the SIMOTION ST Programming and Operating Manual.

Example of "always" initialize:

```
VAR
      {BlockInit_OnDeviceRun := ALWAYS;}
      Test_1 : REAL;
      Test_2 : REAL;
  END_VAR
```

Example of "never" initialize (even if, in the device settings, the option "Initialization of the non-RETAIN global variables (VAR_GLOBAL and global device variables) and program variables (VAR) at the transition from STOP to RUN " is active):

```
  VAR_GLOBAL
      {BlockInit_OnDeviceRun := DISABLE;}
      Test_1 : REAL;
      Test_2 : REAL;
  END_VAR
```

Always / never initialize data in MCC or LAD/FBD sources

In MCC or LAD/FBD sources and/or the associated programs/charts, the initialization during a STOP - RUN transition can be influenced by inserting a pragma line in the declaration table. Click the "Pragmas" button and select the appropriate setting under "Initialization ... at the transition from STOP to RUN".

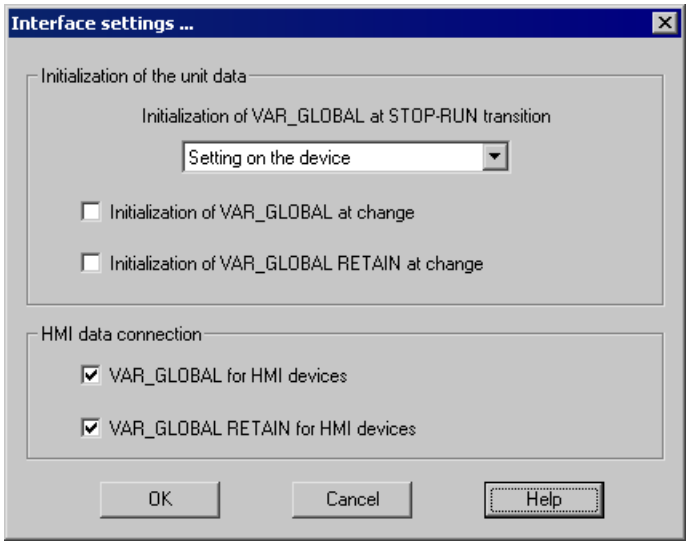


Figure 12-7 Pragma settings for unit variables (interface section)

The default value of "Setting on the device" means:

- As of V4.2: The setting made on the SIMOTION device applies.
- Up to V4.1: Variables are not initialized.

See also section "Adding pragma lines in variable definition" in the MCC Programming and Operating Manual or in the LAD/FBD Programming and Operating Manual.

12.10.7 Download in RUN before programs in Motion Tasks

Description

If you have made changes to sources and want to reload these using download in RUN, an active MotionTask may prevent a change-over time from being found for the interchange of the sources.

Make provision for an operator function (such as single clock mode) so that you can reset one MotionTask, several MotionTasks, or all MotionTasks.
MotionTasks can also be reset via SCOUT as of V4.1.2.

12.10.7.1 Manage MotionTasks via SCOUT (as of V4.1.2)

Description

You can control MotionTasks from SCOUT even without having created a user program.

In this way you can test programs and directly affect the execution of MotionTasks. To perform a download in RUN, you can terminate MotionTasks individually via SCOUT and restart them after download.

12.10.7.2 Switching on debug mode and controlling MotionTasks

Requirement

To control MotionTasks, you must be ONLINE and switch to debug mode.

1. Right-click the relevant device and execute **Operating mode** in the context menu.
2. Then select the **Debug mode** option in the **Operating mode** dialog, read the safety instructions, accept these by clicking the option and then clicking **OK**.

Procedure

1. After you have activated the debug mode, call up the device diagnostics via **Target system > Device diagnostics**.
2. Select a MotionTask on the **Task status tab** and right-click it.
   A context menu is displayed.

3. Select **Disable task start** if you want to block the start of the MotionTask. This prevents the task from restarting (TASKSTART_LOCKED in task status).
4. Select **Reset task** if you want to set the task in the STOPPED state. A download in RUN can be performed in this state.
5. Select **Enable task start** if you want to release the start of the task. You can then restart the task from the program or via SCOUT.
6. Select **Start task** if you want to start the MotionTask.

After executing the task control commands, it takes some time before the display in SCOUT is refreshed. You can also click the **Update (F5)** button.

Controlling several MotionTasks simultaneously

1. Select the desired tasks by holding down the Ctrl key and clicking the left mouse button. The tasks are selected.
2. Click the **Control MotionTasks** button and select the corresponding command in the context menu that appears.

Note

The task control commands depend on the system task control commands. You can read the task status in the **Task runtimes** tab under **Task status**.

Behavior when changing the operating mode

If all tasks are enabled when leaving the debug mode, the CPU retains its operating state.

The system displays the following behavior if you intentionally or unintentionally exit the **Debug mode** operating mode and at least one task is disabled.

- If you exit the debug mode before all tasks are enabled, the CPU switches to STOP and the disabling is canceled.
- If you go OFFLINE before all tasks are enabled, the CPU goes into STOP and the disables are cancelled. If you go ONLINE, the CPU switches to process mode again (debug mode is canceled).
- A RUN - STOP - RUN transition does not affect the task status. If the task was disabled before the start, it is also disabled for the start.
- The CPU goes into STOP if SCOUT unintentionally loses communication to the controller. SCOUT goes OFFLINE at the same time.

12.10.8 Download without HW Config information

Download with inconsistent HW Config

As of V4.1.2, it is also possible to perform a download with inconsistent HW Config. In the event of significant changes to the hardware configuration, the download is still canceled and the HW Config also has to be loaded.

The following parameter changes require a download with HW Config

- Logical addresses
  - Address moved in HW Config
  - Adding separate slots to DP nodes or to existing nodes
- Telegram length
  - Telegram length changed
- Cycle clock settings
  - DP cycle time and PN send cycle clock changed

Note

The settings for the telegram length, address and data types are adapted in HW Config in the **DP Slave Properties** dialog box in the **Configuration** tab.
The cycle clock settings are made in the **Isochronous operation** tab.

Setting download without HW Config

You can select this additional option in **Additional CPU options** in the **Load to target system** and **Load to CPU/drive unit** dialogs. You can preset the additional option in **Options > Settings > Download**.

12.10.9 Download of changed technology objects

Description

You can reload changes in the configuration data and system variables of a TO you made offline in RUN (using restart and effective immediately). If necessary, a TO restart is performed after the download.

Downloading TOs

It is possible to download a changed TO under the following circumstances or prerequisites:

- Only technology objects whose structure is retained can be downloaded (see below)
- A download is also performed when a technology object is used by a program, e.g. axis is enabled. If necessary, TO alarms are then output.
- The substitute values of the TO supplied during the download are returned according to the setting of "restart.behaviorInvalidSysvarAccess" (LAST_VALUE, DEFAULT_VALUE). If a restart is performed during a download of TOs, access errors from the user program may occur. The reaction of the CPU or the return value then depends on the settings of the configuration data item "restart.behaviorInvalidSysvarAccess". If this configuration data item is set to STOPPED, the CPU goes to STOP. For more information, refer to System variables (Page 143).

- If the change to a TO cannot be loaded during the download or an error occurs, the original configuration is retained or re-established (situation as prior to the download).
- The technology objects are loaded in succession and take effect immediately.

Behavior when loading multiple TOs

If you are loading a group of TOs and one of the TOs cannot be downloaded:

- The configuration for already loaded TOs remains active
- The download for the technology object that could not be loaded is rejected.
- The download for the TOs not yet loaded continues

Permitting a TO restart

Before the download is performed, you can use an option to select whether a RESTART of the TOs is to be permitted or not. In this way, you can directly influence a TO RESTART. A RESTART is then executed independently of the setting in the TO in configuration data element RestartCondition.

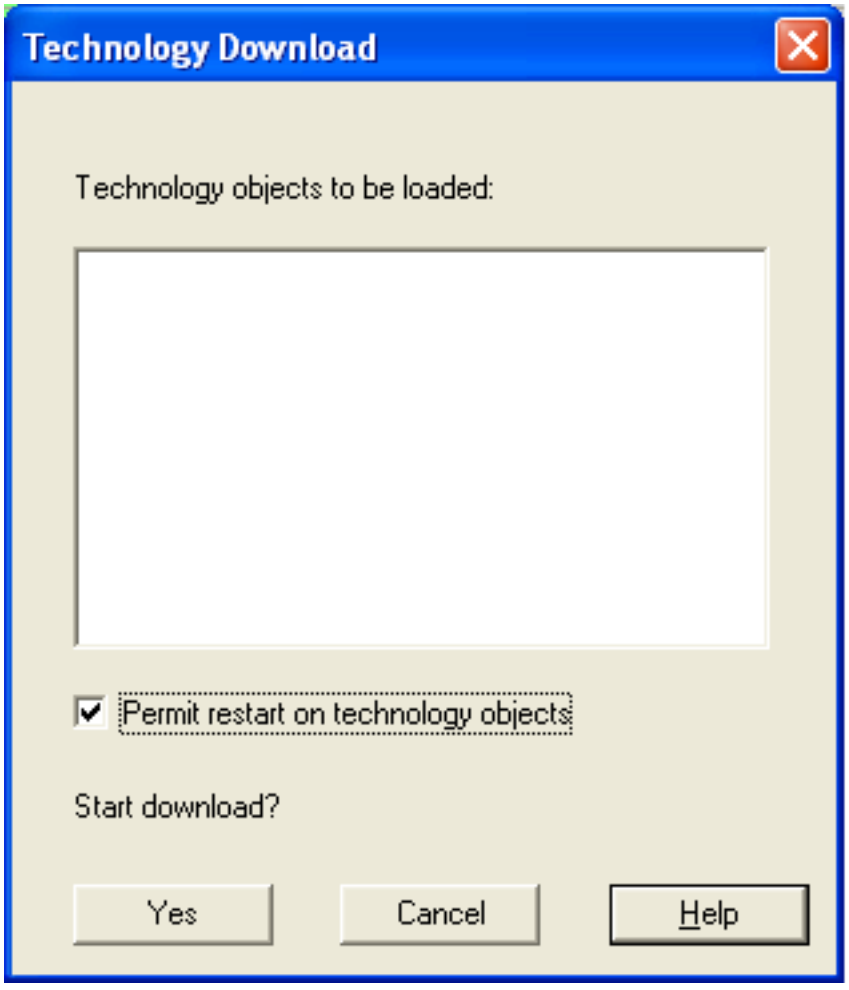


Figure 12-8 TO overview for download

The **Permit restart on technology objects** option is selected by default. If you deselect the option, this setting is saved for further downloads with this project. With configuration data that can only be changed via restart, the TO then behaves as though the download would not have functioned.

Changeable parameters that allow a download in RUN

Configuration data that can be changed via a download always acts on the structure of the TO. In addition, changes to units, interconnections, alarm configuration, and profiles also act on the structure of a TO in such a way that a download in RUN is no longer possible.

In view of this, you are able to change the following data for a download in RUN:

- Immediately effective configuration data
- Configuration data effective at a restart
- System variables

In the expert list of the TOs, the **Effectiveness** column allows you to recognize whether the change in a configuration data item takes effect immediately, requires a restart or is possible only with a download (in STOP). This applies to the configuration data in ONLINE as well as OFFLINE mode.

12.10.10 Copying RAM to ROM in RUN (as of V4.2)

Description

If you have performed a download in RUN, you can also save the changes on the memory card in a powerfail-safe manner.

The Copy RAM to ROM option can be selected beforehand in the download dialog. After a download, this is also possible by means of a separate function (using the **Copy RAM to ROM** button in the toolbar or in the device context menu under Target device).

To preset the option in the download dialog, select **Options > Settings > CPU download**. Here, it can also be specified whether or not the actual values of the TO configuration data are transferred to RAM before the copy RAM to ROM operation.

Note

During "Copy RAM to ROM in RUN", CPU utilization must not be at a critical level (see device diagnostics) as this function generates additional load.

Download with RAM2ROM in RUN in projects with older runtime versions (< V4.1) is not possible and is actively prevented.

Inconsistent TO after a download in RUN

If a TO configuration data element has been changed in the application, it may be the case that the corresponding TO is displayed as inconsistent after a download in RUN.

Scenario with the following settings:

- **Download > Copy RAM to ROM**
- **Settings > CPU download > Accept actual values when copying RAM to ROM**

The configuration data changed by the application will then be in the current data memory. The above settings ensure they are copied from current memory to the RAM and then to the memory card (ROM).
The configured values in the project and the values in the RAM are therefore different and the TO is displayed as inconsistent in the project navigator.

You will then be able to proceed as follows:

- Ignore TO inconsistency
- Analyze the TO inconsistency by means of a project comparison.
- Load the online TO configuration data to the programming device using **Load to PG**.
- Load the offline TO configuration data by performing a download to the device.

See also Copy RAM to ROM (Page 589).

12.11 Download direct to memory card or hard disk

With **Load to file system**, you can save the executable project data from SCOUT to the hard disk of the PC/PG or directly to the memory card using a card adapter.

Two methods of saving are available:

- With the "Save normally" option, the user directory is created containing the SIMOTION and SINAMICS subdirectories.
- The "Save compressed" option creates the same content in the form of a Zip file that can be used, for example, for loading via SIMOTION IT Diag.

The loading to the memory card can be viewed the same as a download with subsequent copying of RAM to ROM. This function is very useful, for example, during series commissioning.

Note that the non-volatile data saved on the memory card are overwritten with **Load to file system**. If required, back up the non-volatile data first as described in Memory access (Page 550).

Procedure

To save the data, SCOUT must be in OFFLINE mode and the device must have been selected in the project navigator for this function to be available.

- Use **Edit > Load to file system** or **Load to file system** in the context menu of the CPU to save the data of a device to a memory card/CF card/SD card or locally to a hard disk.

SCOUT does not automatically recognize the connected read and storage media automatically. The storage/read device must be explicitly selected using **Select target**. An executable project is produced when the project data are saved.

Note

When you save data (either to hard disk or CF card), a USER directory is created automatically so that you can copy data directly to the ROOT directory of a CF card.

⚠ CAUTION

Data loss on the memory card

If you perform **Load to file system**, then the **OEM** folder on the memory card is also deleted. These data are not restored for OEM applications that are not known in the engineering system (are not installed there). You must copy the data manually to the card.

Loading data from the hard disk to the card

You can copy the data from the hard disk to the CF card. As of V4.2, only the contents of the following directories are deleted or overwritten:

- \USER\SIMOTION\RT_DIR
  The subdirectory UPP can be backed up as of version 4.4.
  To do this, check the box "Restore the retentive unit global variables on the device". If the program sources with the compiler option "Permit OPC-UA / -XML (load symbols to RT)" have been compiled, retentive data can be restored by this method. See section "Retaining retentive data" in the "Basic Functions for Modular Machines" Function Manual.
- \USER\SINAMICS\DATA\RT_DIR

The user data in the directories listed below are retained.

Examples of user data saved on the card:

- Backing up retain data (non-volatile data saved using _savePersistentMemoryData) stored in the directory:
  - user\simotion\pmemory.xml
- Backing up IT DIAG user files, settings (e.g. trace.xml), task trace data, log files, and Java files (classes, archives, user file system, etc.), stored in the following directories:
  - user\simotion\hmicfg
  - user\simotion\hmi
- Backing up unit data (data saved on the CF card using _saveUnitDataSet/ _exportUnitDataSet), stored in the following directory:
  - user\simotion\user dir\<unitname>

Note

The stored data must match the firmware of the card. If, for example, you load an old configuration to a card with the latest firmware, you receive error messages during ramp-up.

12.12 Downloading using the device update tool

Description

SIMOTION offers a convenient solution when updating SIMOTION devices or SIMOTION projects for machine manufacturers and machine operators. Upgrading does not simply mean updating to a higher firmware version; it is a general term for switching to a defined configuration, e.g. a project update. It is also possible to return (restore) to a previous configuration. Update or restore procedures can easily be performed on SIMOTION devices in situ or remotely. The data can be imported to a SIMOTION device via a portable, handy storage medium (e.g. memory stick) or a communication connection.

Note

The firmware of connected SINAMICS drives will only be updated if the automatic firmware update function has been set in parameter p7826 (on every drive unit to be updated).

Detailed information can be found under Updating SIMOTION devices.

12.13 Retaining retentive data during download

Retentive unit variables and retentive local variables are retained when a download is performed provided that no changes are made to the data blocks.

Please also note the relevant information in section "Retaining retentive data" in the "Modular Machines" Function Manual.

Reorganization of the retentive memory

If the size of the retentive data blocks changes in the new configuration, fragmentation of the retentive memory may make it impossible to perform the download.

The following applies as of version V4.5 of the SIMOTION Kernel:

- The device must be restarted manually
- The retentive memory will then be reorganized.
  The values of the retentive global unit variables are retained if the symbol information is saved in the device. For this purpose, the compiler option "Permit OPC-UA / -XML (load symbols to RT)" must be activated for all relevant program sources during compilation.
- Please also note the relevant information in section "Retaining retentive data" in the "Modular Machines" Function Manual.

The following applies up to version V4.4 of the SIMOTION Kernel:

- You must perform a memory reset on the device.
  The retentive data is lost.

12.14 Loading data from the target device to the programming device (PG)/PC

With **Load to PG**, the data is loaded from the target device to the PG/PC. In this way, the currently loaded project can be loaded from the target device with all settings to the local hard disk.

Sources or additional data of a project can only be loaded to a programming device when they have previously been loaded to the target system. **Options > Settings > Download > Store additional data on the target device** must have been set in the project for this.

The menu item **Load to PG** is only active when you are ONLINE and a device has been selected.

Note

This function is not suited to restore the original project in SIMOTION SCOUT. Use the **Target system > Load > Copy archived project from card to PG/PC** function for this purpose.

The function is more suited to restore an archived original project, see the following Section "Loading with an existing original project".

Further information on the archiving of a project and saving of the archived project on the memory card can be found in the Online help.

Load to PG with available original project (maintenance scenario)

You are placing commissioned equipment into service, along with a project, or loading the archived project from the memory card of the controller. The project is not consistent online, but is essentially the same as the project that will run on the target device. You can use **Load to PG** when performing the upload to create online consistency between the SCOUT project and the CPU. Afterwards, possible errors are searched for and changes downloaded to the target device.

Note

Objects in the OFFLINE project will be overwritten. As a result, objects which are only present OFFLINE will be deleted. If you want to still be able to access the OFFLINE project after the upload, click **Save before loading to PG** to save the current version of the OFFLINE project. You can retain the OFFLINE project again by exiting the project after the upload without saving it.

Alternative: Archive the OFFLINE project first or save it under another name.

Please note that various functions (e.g. program status) remain inactive and cannot be used until the uploaded objects are saved with the project.

The offline and online projects can also be aligned using the comparison functions (project comparison).

Procedure where there is an existing project

1. Perform **Target system > Load > Load CPU / drive unit to PG**.
   The **Load to PG** dialog box is displayed.
2. Select the option **Load target device to PG** if you want to load all the project data from the target device to the programming device. Note the information in the dialog box.

3. Select the option **Save before loading to PG** if you want to save the project on your PG before uploading.
4. Select the option **Overwrite available libraries** if you want to overwrite the libraries of the project available locally on your programming device.

Only load configuration data to PG

1. Select the **Only load configuration data to PG** option if you want to load the configuration data from the RAM to the programming device.

Transfer current data to RAM

If you want to load the amended configuration data in the current data memory of the target device to the RAM before loading to the programming device, select the **Transfer current data to RAM** checkbox. If you do not transfer the current data to the RAM, it is not loaded to the programming device.

As of V4.2, configuration data is completed during ramp-up using parameter values of the SINAMICS drive (adaptation). For more detailed information, see **Adaptation** under Symbolic assignment - introduction (Page 87). When "Load to PG" is executed, adapted values are detected and "Transfer current data to RAM" is automatically preselected as well.

Loading to the programming device via a project comparison (object granular)

- With the project comparison functionality, you can also load the data for individual objects to the programming device. For more detailed information, please refer to the online help on project comparison or the sections relating to project comparison in the SIMOTION manual.

See also

Overview of the data download (Page 555)

12.15 Copy current data to RAM

Description

Use **Copy current data to RAM** to copy the current configuration data, changed during the RUN mode, to the RAM. Depending on the data item, changes to the configuration data during RUN take effect in different ways (e.g. immediately or after a restart of the technology object).

After the restart on the technology object, the changed configuration data is stored in the current data memory. To copy the values from the current data memory to the RAM, you must execute this function explicitly. Only after copying is the data in the RAM, and is then

uploaded at the next ramp-up and takes effect. See also Overview of the memory in the target device (Page 547).

Note

So that the project data in SCOUT is consistent with the project data in the target system if configuration data has been changed online, you must perform an upload to the PG/PC (menu **Target system > Load > Configuration data to PG**).

Adaptation of configuration data

When SIMOTION devices ramp up, reference variables, along with drive and encoder data for SINAMICS S120 (SINAMICS Integrated), are transferred automatically for the configuration data of the SIMOTION technology objects "TO axis" and "TO external encoder". Therefore, this data no longer has to be entered in SIMOTION. The adapted data is located in the current data memory. When **Copy current data to RAM** is executed, these values are added to the RAM. This makes the technology object inconsistent online. To resolve this inconsistency, you also need to load the configuration data to the PG. This is detected in the dialog and the appropriate preselections are made. If you have changed configuration data online yourself, this will also be loaded to the PG. For more detailed information on adaptation, see Symbolic assignment - introduction (Page 87).

Transferring current data to RAM

1. Select **Target system > Copy current data to RAM**.
   The **Copy current data to RAM** dialog opens. The dialog indicates whether the configuration data has been changed as a result of adaptation.
2. Select one of the following options:
   – Copy RAM to ROM
   – Load to PG
3. Click **Yes** to trigger the copying process and any options selected.

For more setting options to trigger **Copy current data to RAM** select:

- **Load to PG** dialog
- **Options > CPU download >Copy RAM to ROM > Transfer current values to RAM**

See also

Copy RAM to ROM (Page 589)

12.16 Copy RAM to ROM

Description

Select **Copy RAM to ROM** to save the project from the volatile memory (RAM) to the retentive memory (ROM) of the memory card. A power failure must be avoided when writing to the memory card.

For a detailed description, see Overview of the memory in the target device (Page 547).

Possible errors when copying from RAM to ROM

Errors can occur when copying from RAM to ROM Possible causes:

- An access error has occurred.
- Rotating measurement (p1960) is activated.
- Insufficient storage space is available. Check the memory card and restart the action.
- The ramp up of the drive unit has not yet completed, check the operating state of the diagnostic overview.

Note

Check the entries in the diagnostic buffer and the **target system output**.

Copying RAM to ROM with adapted configuration data

When SIMOTION devices ramp up, reference variables, along with drive and encoder data for SINAMICS S120 (SINAMICS Integrated), are transferred automatically for the configuration data of the SIMOTION technology objects "TO axis" and "TO external encoder". Therefore, this data no longer has to be entered in SIMOTION. The adapted data is located in the current data memory, **not** in the RAM.

When **Copy RAM to ROM** is executed, the system detects whether values have been adapted. **Copy current data to RAM** and **Load configuration data to the PG** are also preselected in order to back up this configuration data to the memory card (ROM) and to carry out alignment with the engineering project as well. See also Copy current data to RAM (Page 587).

See also

Copying RAM to ROM in RUN (as of V4.2) (Page 582)

12.17 Deleting user data on the memory card

You will need to delete the user data on the memory card, for example:

- If you want to save a different (new) project to the memory card and, therefore, delete any user data relating to an "old project" (e.g. unit data sets) which might be stored on the memory card.
- If you want to delete the technology packages on the memory card, e.g.:
  - If you want to use a smaller technology package (e.g. CAM instead of CAM_EXT)
  - To force an upgrade of the technology package **within a version** in line with a new hotfix or service pack

  Note

  User data does not have to be deleted in the event of a change of version. In such cases, the technology packages on the memory card are always updated.

- You can delete the user data with SIMOTION SCOUT. As of V4.2, you can select which data you want to delete.

This means you can still access the SIMOTION module online with your PG/PC. The licenses on the memory card are retained.

Deleting user data

1. In SIMOTION SCOUT, open the project you want to edit.
2. Go online with the module.
3. Select the module in the project navigator and then select **Target system > Delete user data on card**.
   The **Delete user data on card** dialog appears. You can select which data you want to delete in this dialog:
   - Project data (programs, technology objects, TPs, SINAMICS Integrated, etc.)
   - Unit data sets
   - Non-volatile data (overall reset)
   - Archived project
4. Activate the options which you want to delete.
5. Click **OK** to delete the data.

   Note

   Non-volatile data and unit data sets may only be deleted together with project data.

Error sources and efficient programming 13

13.1 Error sources during programming

13.1.1 Error sources during programming

Explanations of the most important error sources associated with programming are provided below. The section also provides you with solution possibilities for eliminating these error sources.

13.1.2 Checking data types when assigning arithmetic expressions

In arithmetic expressions, the result is always calculated in the largest number format contained in the expression.

An expression value can only be assigned to a variable in the following cases:

- The calculated expression and the variable to be assigned are of the same data type.
- The data type of the calculated expression can be implicitly converted to the data type of the variable to be assigned.

Therefore, if you want to assign an expression to a variable:

- Ensure that the variable is of a large enough data type.
- Perform an explicit data type conversion for that part of the expression that is too large (see Functions for the conversion of numeric data types and bit data types (Page 387)). However, as a result information could be lost, for example, if the value range is decreased or the accuracy is reduced, as is the case for conversion from LREAL to REAL.
- If applicable, specify the data type explicitly for numbers (e.g. UINT#127, if the number 127 is to be of data type UINT instead of USINT).

13.1.3 Checking start of functions in cyclic tasks every time

Technology object functions, such as functions for positioning axes, should only be started in cyclic tasks if they are not already running. Otherwise, the commands in your program are executed every time the cycle is repeated.

You can program a condition for the technology object function start in cyclic tasks, e.g. depending on the contents of an auxiliary variable, which is set when the command is executed.

Table 13-1 Example for correct TO function start in a cyclic task

```
//...
IF myStart = 0 THEN       // If auxiliary variable has not yet been set
    myStart := 1;        // Set auxiliary variable (function started)
    myCommandID := _getCommandId ();
    myFC := _pos (axis := myAxis, // Execute function
             position        := position_1,
             nextCommand     := IMMEDIATELY,
             commandID       := myCommandID);
END_IF;
//...
IF myAxis.positioningState.actualPosition = position_1 THEN
    myStart := 0;         // Reset auxiliary variable, if
                          // function execution required.
END_IF;
//...
```

13.1.4 Wait times in cyclic tasks

If you link command transitions to conditions when using system functions in cyclic tasks, e.g. for the *_pos* command, a timeout can occur, causing a CPU stop.

This can occur in any system function where the *nextCommand* parameter assumes a value other than IMMEDIATELY, e.g. the value WHEN_MOTION_DONE.

The timeout occurs if the cycle time configured in SIMOTION SCOUT is exceeded as a result of a step enabling condition or programmed wait times, e.g. using *_waitTime*.

Note

Do not use commands with wait times in cyclic tasks, e.g. *_waitTime*.

Use only input parameter **nextCommand := IMMEDIATELY** for system functions in cyclic tasks.

13.1.5 Using the commandId parameter correctly

All TO commands must contain a parameter for command identification, see **Function parameters of technology functions.**

Prior to calling the appropriate command you can obtain a project-wide unique command ID with the command *_getCommandId*. Save the command ID in a local variable and use

it as a parameter in the technology object command; alternatively, use the function call in *commandId:=_getCommandId()* directly as a parameter.

Note

If the CommandId is not assigned for the command call, the default value of the parameter takes effect with (0.0).

The CommandID is not only used by TO commands.

You must use this unique command ID to check the status of the motion command, e.g. to check the status of a positioning motion with *_getStateOfAxisCommand*. The system can only uniquely identify a motion command from its command ID!

Table 13-2 Example of using a TO function with command identification

```
//...
VAR
    myCommandID                 : commandIdType;
    myState                     : StructRetCommandState;
END_VAR
//...
myCommandID := _getCommandId ();
// Save unique ID
myFC := _pos (axis := myAxis,
// Execute function with ID
             position           := position_1,
             nextCommand     := IMMEDIATELY,
             commandID          := myCommandID);
myState := _getStateOfAxisCommand (axis:=myAxis,
             commandID          := myCommandID);
// Status check
IF myState.commandIdState = WAITING_FOR_SYNC_START THEN
    ;
//...
END_IF;
//...
```

See also

Function parameters of the technology functions (Page 126)

_getCommandId function (Page 448)

_getSyncCommandId function (Page 448)

CommandID overview (Page 447)

13.1.6 Locating errors (ST programs)

The following error message can occur during compilation:

Error 7000, 7010, 7011, or 7014 in Line ...

A syntax error has occurred. Possible causes:

- *Incorrectly terminated control structures (e.g. END_IF missing)*
- *Statements not terminated with ;*
- *Missing parentheses*

First, check the specified line to see whether the error actually occurred there, i.e. perhaps you have not terminated a control structure, or have omitted a semicolon at the end of a line, or a parenthesis is missing.

If you cannot find any of the specified errors, you must locate the error:

1. Comment out the program block-by-block before the line containing the error message, i.e. enclose the selected section between the character pair (* and *). The line with the error message must not be commented out.
2. Recompile the program.
3. If an error message still appears after steps 1 and 2, the commented-out section is too small. Expand it until the error message disappears.
4. When the error message no longer appears, the error is contained in the commented-out section. Now reduce the size of the commented-out section line by line and recompile the program each time until the error message appears again. The last enabled line is the line with the error message.

Note

You can consult a list of all compiler error messages in the appendix of the ST Programming Manual.

13.1.7 Errors on download

If an error message occurs when your program is downloaded, the log is held in the SIMOTION SCOUT detail view. Look for the error cause in the log. Check your hardware configuration or the program, e.g. for addresses that do not exist.

For more information about hardware configuration and addressing, refer to the SIMOTION SCOUT configuration manual.

Error messages during the download that contain the abbreviation UPP (User Program Processing) occur when a user program is being processed, e.g. when changes cannot be performed in RUN.

13.1.8 CPU does not switch to RUN

If the CPU switches back to STOP mode immediately after you have started your programs, check the device diagnostics and alarm window in SCOUT. The causes of the STOP are entered in the diagnostics buffer. The alarms of the technology objects that can also cause a CPU STOP are displayed in the Alarms window.

13.1.9 CPU goes to STOP

Description

If the CPU changes to STOP mode then check the device diagnostics and the alarm window in the SCOUT. The causes of the STOP are entered in the diagnostics buffer. The alarms of the technology objects that can also cause a CPU STOP are displayed in the Alarms window.

Possible reasons, why a CPU can go into STOP are:

- Incorrect direct I/O access
- Configuration data access or system variable access to TO if in RESTART (as of V4.1, however, substitute values for system variables are possible; see System variables (Page 143))
- Missing PeripheralFaultTask (if there is incorrect access to the process image)
- Missing TechnologicalFaultTask (if there are errors on the technology object)
- Processing error in the programs (or missing ExecutionFaultTask)
- Monitoring time overflow of the IPO/servo tasks
- Monitoring time overflow of the BackgroundTask
- Monitoring overflow of a TimerInterruptTask
- Sign-of-life monitoring Simotion - drive
- Technology object alarms with STOP response (e.g. 20001)
- Set mode using system variable

Displaying and changing the operating mode

Use the device system variable *modeOfOperation* to display or change the current operating mode.

Syntax example

```
modeOfOperation  :EnumDeviceModeOfOperation  //readable, writable,
immediately effective
EnumDeviceModeOfOperation:
    [
     _STOP |
     _RUN
    ]
```

For example, with this, you can switch a SIMOTION CPU that has gone into STOP into RUN again from a local HMI by writing the device system variables.

Detailed information about the system variable *modeOfOperation* can be found in the *System Functions/Variables of the Devices* Manual or in the online help.

See also

Execution errors in programs (Page 154)

Access errors to system variables and configuration data, as well as I/O variables for direct access (Page 156)

SystemInterruptTasks (Page 232)

Possible errors in technology objects (Page 179)

13.1.10 Checking and setting system clocks

Often, incorrectly set system cycle clocks (servo cycle clocks, interpolator cycle clocks, PWM cycle clock) cause the SIMOTION device to go into STOP mode.

Check the runtimes of the SynchronousTasks (ServoSynchronousTask, ServoSynchronousTask_fast, IPOsynchronousTask, IPOsynchronousTask_fast, IPOsynchronousTask_2, tasks for the TControl technology package). It could be that the user programs or system programs for individual tasks need more time than is set in the system cycle clocks in SIMOTION SCOUT.

Also try to minimize the runtime of the SynchronousTasks. Shift the programs to MotionTasks as much as possible, and split up your programs accordingly, if necessary.

Make sure there is an integer ratio between individual tasks. Otherwise, low-priority SynchronousTasks will not be started at periodic intervals.

Deactivate unnecessary tasks.

Note

The system tasks for the TControl technology package can be disabled in the system cycle clock settings window.

Refer to the online help for information about how to check the device diagnostics, interpret the alarm window and check/change your system clocks.

System functions and a Task Trace are available for checking the runtimes. The Task Trace can be used to graphically display the sequence of individual tasks and user events (generated per program command); see Task Trace function manual.

See also Isochronous data processing (Page 285) , Functions for message programming (AlarmS) (Page 356).

13.1.11 Comparing REAL or LREAL variables

When you compare REAL variables or LREAL variables (also corresponding system variables, e.g. axis position) with one another, you should never use "=". Because of the different internal number representation, the compared numbers are never identical. Instead, you should evaluate, for example, the direction of motion and use ">" or "<" or the system variable for "Position window reached".

13.1.12 Sequential task is interrupted

Description

Sequential tasks (MotionTasks) can assume different states, including TASK_STATE_WAITING. The status can be read out in the diagnostics display of the CPU.

If you do not know why the task is waiting, you must carry out an extensive search to identify the point in the program at which the task is waiting on a condition.

You use the following functions to find these points:

- Display program run function
- Display code position (e.g. line of an ST source file) that runs a MotionTask

For detailed information refer to Section *Program run* in the *ST programming manual*.

13.1.13 Checking for range violations

Range violations, i.e. the violation of range limits for a data type, are not signaled by the compiler. Therefore, if you use variables in arithmetic operations, you should always check for possible range violations.

Table 13-3 Example of checking for a range violation

```
PROGRAM myRange
    VAR
        a,b  : SINT := 100;
        c: SINT;
    END_VAR

    c := a + b;
    // If c is outside range, then exit program.
    IF (a > 0) AND (c < a) AND (c < b) OR
       (a < 0) AND (c > a) AND (c > b) THEN
       // Range violation
       RETURN;
    ELSE
        ; // OK
    END_IF;
END_PROGRAM
```

13.1.14 Setting size of local data stack

When configuring the execution system, set the size of the reserved local data stack for each task in the *Task configuration* tab, see Description of the respective user program task (Page 212).

Use the "Program structure" function to obtain information about the memory requirements of a program with all called program organization units (FCs and FBs) on the stack, see

the SIMOTION ST Programming Manual. You will also find information on the memory requirement of variables on the local data stack in this manual.

During configuration, ensure that there is sufficient spare. For example, temporary additional storage may be required on the local data stack during downloading in RUN.

The download operation will be aborted with an error message if the reserved local data stack exceeds the available memory space in the RAM.

The program will be aborted during execution if the reserved local data stack is insufficient for program execution.

In the event of an error, check the following:

- RAM utilization in the device diagnostics
- The sources (programs, etc.) for large data structures (arrays, structures), e.g. using the cross-reference list
- Size of local data stack for the task

See also

Specifications for the configuring (Page 320)

13.1.15 Sources of errors during multitasking

The main sources of errors during multitasking are:

- Using FB instances in different tasks
- Using global variables in different tasks

Using FB instances and global variables in different tasks carries the risk of this data not being processed correctly. If global variables are used in different tasks, a change of task and processing in a different task may cause an unwanted change in the value, for example.

13.2 Efficient programming

13.2.1 Efficient programming - overview

The following discrepancy always occurs in many control systems and/or the associated programming environments:

- Well structured and concise user programs with special emphasis on modifiability and expandability do not perform optimally with regard to runtime.
- On the other hand, runtime-optimized programs are difficult to expand or modify.

The following description provides information for runtime-optimizing programming and for change-optimizing programming. Depending on the task or with local optimizations, you should focus on one of these options.

13.2.2 Runtime-optimized programming

The following description contains information about runtime-optimized programming. Note that instructions for runtime-optimizing programming are often at odds with the rules of structured programming.

13.2.2.1 Optimizing access to inputs and outputs

Access to the process image of cyclic tasks is significantly faster than direct access to inputs or outputs (see Direct access and process image of the cyclic tasks in the ST Programming Manual). Therefore, you should assign an I/O variable to the process image of the task in which the variable is used. In addition to quicker access, another advantage of this is that the I/O variable will be consistent during the entire runtime of the task.

13.2.2.2 Optimizing access to system variables

It takes considerably longer to access system variables (see System variables (Page 143)) than to access variables that are stored in the dynamic memory (local variables, non-retentive unit variables).

Therefore, only a few instances of direct access to system variables are possible in high-speed cyclic tasks (e.g. SynchronousTasks). For this reason, when a system variable is accessed many times, it is recommended that you copy the entire system variable structure at the beginning of a cycle (program in the cyclic task) to a local variable of the corresponding system data type. Then access this variable during the program cycle.

The data types for declaring the local variables can be found in the List Manuals for SIMOTION technology objects (TOs).

13.2.2.3 Optimal variable declaration

Arrange the variables within a declaration block (e.g. VAR/END_VAR) in order of ascending size. This enables you to make optimal use of the memory space.

Initialize variables with values other than 0 only when necessary. Variable initialization at the start of a task or POU requires time. This is especially the case for temporary variables and variables of programs that are assigned to sequential tasks.

13.2.2.4 Optimizing access to function block parameters

When creating function blocks (FBs) that are to process values, you can select one of two methods. You can assign input variables in the FB with VAR_INPUT, process the variables there, and assign the results to output parameters with VAR_OUTPUT.

If a large data volume is to be transferred to the FB, it can be faster to use in/out parameters (VAR_IN_OUT) than to use input and output parameters (VAR_INPUT and VAR_OUTPUT).

Parameters can be transferred more quickly because copying is eliminated, however, it could take longer to access the variable from the FB under certain conditions.

Note

When you access unit variables or global device variables with in/out parameters, note that other tasks can access these variables at the same time.

13.2.2.5 Optimizing program structure

Make sure that your ST source files and the POUs they contain are structured clearly and concisely. However, avoid modularizing your source files too much, since it takes longer to access functions and variables of imported units than functions and variables within a unit.

13.2.2.6 Optimizing the execution system

Assign the IPOsynchronousTask to a single program only. This reduces the risk of a timeout during runtime.

Use functions that are executed synchronously in MotionTasks only. (During synchronous processing, the next command is not executed until processing of the pending command satisfies a condition, e.g. it is complete.)

Make sure that not too many MotionTasks are active simultaneously.

For more efficient programming of cyclic tasks (primarily the BackgroundTask) go to sequence controlled programming. This means you consciously control the program flow via case decisions and only run through the portions of code that are relevant in the current status (e.g. via CASE statement).

Moreover edge triggered program instead of level-triggered programming is recommended.

Do not run through the relevant code cyclically (if the level of the condition is TRUE) but rather just once. This is done by querying the satisfied condition on rising edge (new status of the condition is TRUE, the old status on the other hand was FALSE).

13.2.2.7 Use of USEPACKAGE for multiple technology packages

Syntax for the use of "USEPACKAGE" for several technology packages

If you want to use more than one technology package simultaneously, you can select it again using the following syntax. See also Using technology functions in a program (Page 121).

Table 13-4 Formal description

```
usepackagestatement ::= 'USEPACKAGE' packagelist ';'
packagelist         ::= packageentry {',' packageentry }
packageentry        ::= simplepackname | namespacepackname
simplepackname      ::= packagename
namespacepackname ::= packagename 'AS' namespaceident
//packagename and namespaceident must be valid identifiers
```

13.2.3 Change-optimized programming

The following description contains information about change-optimized programming.

For information about downloading in RUN, please see Download of changed sources in RUN (Page 568) .

13.2.3.1 Notes on the draft program

Description

Useful notes on the following, for example:

- How best to structure the program
- How best to use the different tasks (MotionTask, BackgroundTask, IPO task, etc.)
- How best to divide your program between the different tasks

can be found in the application guidelines for SIMOTION.

You can find the application guidelines for SIMOTION in **SIMOTION Utilities & Applications**, which is included in the scope of delivery of SIMOTION SCOUT (under **Documentation > Application guidelines for SIMOTION**).

13.2.3.2 Generic programming

Description

Program axes in arrays and loops. This offers advantages, especially when new axes are being added or errors corrected (only once in the loop and not separately for each axis), and results in a more compact code. The readability is significantly improved.

With MCC and LAD/FDB sources, program initialization can also be activated using a pragma line in the declaration tables ("BlockInit_OnChange" pragma).

You can also information on axis arrays under **Documentation > Application guidelines for SIMOTION** in the section titled **Generic programming** in SIMOTION Utilities & Applications, which is part of the SIMOTION SCOUT scope of delivery.

13.2.3.3 Declaring retentive variables in one unit

Declare all retentive variables in the interface section in one single unit. This has the following advantage:

- It enables you to make optimal use of limited memory space.
- The variables are only initialized if the interface section in this unit has been changed.
- As of V4.1 you can also use multiple declaration blocks, see Use multiple VAR_GLOBAL, VAR_GLOBAL RETAIN blocks (Page 322) .

13.2.3.4 HMI variables in one unit

Declare variables that are exported to HMI devices (see HMI (Human Machine Interface) coupling (Page 519)). In the case of larger projects or strictly delimited software modules, a separate HMI unit per module is appropriate.

- The HMI project (e.g. WinCC flexible) must only be downloaded again if the interface section of this unit has been changed.
- You also have the option of disabling the consistency check during commissioning (**Options > Settings > Download**), though you do so at your own risk. Please note that, as there is no check for inconsistencies (e.g. for valid hardware addresses), access to processes may go unmonitored.
- You can also mark HMI relevant data, see Marking HMI relevant data (Page 325).

See also HMI (Human Machine Interface) connection (Page 519).

13.2.3.5 Using device global variables versus unit global variables

Description

Use of global unit variables in sources is preferable to use of device global variables (via the project navigator).

Benefits:

- Variable structures can be used.
- Initialization (initial values) of the variables for the STOP-RUN transition is possible (via program in StartupTask)
- For newly created global unit variables a download in RUN is also possible (in a new declaration block)

Procedure

1. Define unit-global variables in a source file in the interface section.
   Retain variables and HMI variables should likewise each be declared in a separate source file or declaration block (see above).
2. Assign the program with the initialization of the variables to the StartupTask.
   The variables will be set to a defined initial value in the startup.

Parameter | I/O symbols | Structures | Enumerations | Connections

	Structure name	Element name	Data type	Array length	Initial value
1	MyStruct	IntValue	INT		0
2		RealValue	REAL		0.0
3		BitValue	BOOL		FALSE
4					

Figure 13-1 Example of the declaration of structures in an MCC source file

Parameter | I/O symbols | Structures | Enumerations | Connections

	Name	Variable type	Data type	Array length	Initial value
1	m	VAR_GLOBAL CONSTANT	INT		16
2	BitArray	VAR_GLOBAL	BOOL	m	16(False)
3	IntArray	VAR_GLOBAL	INT	m	16(0)
4	ReadArray	VAR_GLOBAL	REAL	m	16(0.0)
5	StructVar	VAR_GLOBAL	MyStruct		
6					

Figure 13-2 Example of the declaration of parameters in an MCC source file

Table 13-5 Example declaration and program in an ST source file (unit)

```
INTERFACE
    //global types
    TYPE
       MyStruct : STRUCT
                   Intvalue : INT;
                   Realvalue: REAL;
                   Bitvalue : BOOL;
                  END_STRUCT
    END_TYPE
    //global constants
    VAR_GLOBAL CONSTANT
        n : INT := 0;
        m: INT := 15;
    END_VAR
    //global variables
    VAR_GLOBAL
        Bitarray : ARRAY [n..m] OF BOOL := [16 (FALSE)];
        Intarray : ARRAY [0..15] OF INT := [16 (0)    ];
        Realarray: ARRAY [0..15] OF REAL:= [16 (0.0)  ];
        StructVar: MyStruct;
    END_VAR
    //programs
    PROGRAM Init;
    //end of the interface
END_INTERFACE

IMPLEMENTATION
 PROGRAM Init
  ////////////////////////////////////////////////
  //initialization of the variables during startup//
  ////////////////////////////////////////////////
  StructVar.Intvalue :=0;
  StructVar.Realvalue:=0;
  StructVar.Bitvalue :=FALSE;
 END_PROGRAM
END_IMPLEMENTATION
```

The program is then assigned to the StartupTask.

Whenever unit global variables are to be used in a different source, the sources that contain the declaration must be combined (USES). See also **Connection to other program sources or to libraries** in the MCC Programming Manual or **Use of the USES statement in the interface or implementation section of an importing unit** in the ST programming manual.

13.2.3.6 Centralized starting and resetting of MotionTasks

To enhance programming clarity, program starting and resetting of MotionTasks in one centralized location.

Appendix

A

A.1 Symbolic constants

The following table presents names of reserved constants that you must not use for individual variable names.

Table A-1 Symbolic constants of the Taskstartinfo

Symbolic constant	Data type	Value
_SC_ALARM_CONFIGURATION	UDINT	404
_SC_ARRAY_BOUND_ERROR_READ	UDINT	502
_SC_ARRAY_BOUND_ERROR_WRITE	UDINT	503
_SC_BACKGROUND_TIMER_OVERFLOW	UDINT	300
_SC_CYCLE_TIMER_OVERFLOW	UDINT	301
_SC_DEVICE_COMMAND	UDINT	401
_SC_DIAGNOSTIC_INTERRUPT	UDINT	201
_SC_DIVISION_BY_ZERO	UDINT	500
_SC_DP_CLOCK_DETECTED	UDINT	207
_SC_DP_SLAVE_NOT_SYNCHRONIZED	UDINT	210
_SC_DP_SLAVE_SYNCHRONIZED	UDINT	209
_SC_DP_SYNCHRONIZATION_LOST	UDINT	208
_SC_DRIVE_OBJECT_FAULT	UDINT	217
_SC_DRIVE_OBJECT_ALARM	UDINT	218
_SC_EXCEPTION	UDINT	403
_SC_EXTERNAL_COMMAND	UDINT	402
_SC_IMAGE_UPDATE_FAILED	UDINT	204
_SC_IMAGE_UPDATE_OK	UDINT	206
_SC_INVALID_ADDRESS	DINT	–1
_SC_INVALID_FLOATING_POINT_OPERATION	UDINT	501
_SC_IO_MODULE_NOT_SYNCHRONIZED	UDINT	215
_SC_IO_MODULE_SYNCHRONIZED	UDINT	214
_SC_MODE_SELECTOR	UDINT	400
_SC_PC_INTERNAL_FAILURE	UDINT	205
_SC_PULL_PLUG_INTERRUPT	UDINT	216
_SC_PROCESS_INTERRUPT	UDINT	200
_SC_STATION_DISCONNECTED	UDINT	202
_SC_STATION_RECONNECTED	UDINT	203
_SC_TO_INSTANCE_NOT_EXISTENT	UDINT	506
_SC_VARIABLE_ACCESS_ERROR_READ	UDINT	504
_SC_VARIABLE_ACCESS_ERROR_WRITE	UDINT	505

Table A-2 Symbolic constants of the task status

Symbolic constant	Data type	Hex notation
TASK_STATE_INVALID	DWORD	16#0000
TASK_STATE_LOCKED	DWORD	16#0100
TASK_STATE_RUNNING	DWORD	16#0004
TASK_STATE_STOP_PENDING	DWORD	16#0001
TASK_STATE_STOPPED	DWORD	16#0002
TASK_STATE_SUSPENDED	DWORD	16#0020
TASK_STATE_WAIT_NEXT_CYCLE	DWORD	16#0040
TASK_STATE_WAIT_NEXT_INTERRUPT	DWORD	16#0080
TASK_STATE_WAITING	DWORD	16#0010

Table A-3 Symbolic constants for alarm messages

Symbolic constant	Data type	Hex notation
ALARMS_ERROR	DWORD	16#8000
ALARMS_QSTATE	DWORD	16#0100
ALARMS_STATE	DWORD	16#0001
DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE	DWORD	16#0006
DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED	DWORD	16#0005
DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID	DWORD	16#0001
DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR	DWORD	16#0009
DSC_SVS_DEVICE_ALARMS_IV_CALL	DWORD	16#0004
DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL	DWORD	16#0007
DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP	DWORD	16#0008
DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED	DWORD	16#0002
DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED	DWORD	16#0003
DSC_SVS_DEVICE_ALARMS_NO_ENTRY	DWORD	16#0010

Table A-4 Symbolic constants for the value range limits of elementary data types

Symbolic constant	Data type	Value	Hex notation
SINT#MIN	SINT	-128	16#80
SINT#MAX	SINT	127	16#7F
INT#MIN	INT	-32768	16#8000
INT#MAX	INT	32767	16#7FFF
DINT#MIN	DINT	-2147483648	16#8000_0000
DINT#MAX	DINT	2147483647	16#7FFF_FFFF
USINT#MIN	USINT	0	16#00
USINT#MAX	USINT	255	16#FF

Symbolic constant	Data type	Value	Hex notation
UINT#MIN	UINT	0	16#0000
UINT#MAX	UINT	65535	16#FFFF
UDINT#MIN	UDINT	0	16#0000_0000
UDINT#MAX	UDINT	4294967295	16#FFFF_FFFF
T#MIN TIME#MIN	TIME	T#0ms	16#0000_0000[1]
T#MAX TIME#MAX	TIME	T#49d_17h_2m_47s_295ms	16#FFFF_FFFF[1]
TOD#MIN TIME_OF_DAY#MIN	TOD	TOD#00:00:00.000	16#0000_0000[1]
TOD#MAX TIME_OF_DAY#MAX	TOD	TOD#23:59:59.999	16#0526_5BFF[1]
[1] Internal display only			

Table A-5 Symbolic constants for invalid values of selected data types

Symbolic constant	Data type	Meaning
STRUCTALARMID#NIL	StructAlarmId	Invalid AlarmId
STRUCTTASKID#NIL	StructTaskId	Invalid TaskId
TO#NIL	ANYOBJECT	Invalid technology object

A.2 Identifiers with defined meaning in SIMOTION

This section contains an alphabetical listing of identifiers with a predefined meaning in SIMOTION. The list contains:

- The reserved and protected identifiers of the SIMOTION ST (Structured Text) programming language
- The reserved and protected identifiers of other programming languages
- The general standard functions and standard function blocks with the associated data types
- The functions for task control, runtime measurement and message programming with the associated data types
- The system functions and system variables of SIMOTION devices with their associated data types
- The technology object data types
- The system functions, system variables (and configuration data) of technology packages

The response when using this identifier is different, e.g.:

- Compiler error with reserved identifiers
- Possible masking and resulting inability to obtain the identifiers
  Under certain circumstances, the compiler may not issue a warning if, for example, the associated technology package is not imported.

Icons

_abortAllReadWriteDriveParameterJobs
_abortReadWriteRecordJobs
_activateConfiguration
_activateDpSlave
_activateDpSlaveAddress
_activateNameOfStation
_activateTo
_adaptAxisConfigData
_adaptExternalEncoderConfigData
_AdditionObjectType
_addPointToCam
_addPolynomialSegmentToCam
_addSegmentToCam
_addSegmentToCam_V2_0
_alarmS
_alarmSc
_alarmScId
_alarmSId
_alarmSq
_alarmSqId
_allocateTokenToVariableName
_AND
_BOOL
_bufferActuatorCommandId
_bufferAdditionObjectCommandId
_bufferAxisCommandId
_bufferAxisCommandId_V3_1
_bufferCamCommandId
_bufferCamTrackCommandId
_bufferControllerObjectCommandId
_bufferExternalEncoderCommandId
_bufferExternalEncoderCommandId_V3_1
_bufferFixedGearCommandId
_bufferFollowingObjectCommandId

_bufferFollowingObjectCommandId_V3_1
_bufferFormulaObjectCommandId
_bufferMeasuringInputCommandId
_bufferOutputCamCommandId
_bufferPathObjectCommandId
_bufferSensorCommandId
_BYTE_FROM_8BOOL
_BYTE_TO_8BOOL
_calculateTControllerParameter
_CamTrackType
_cancelAxisCommand
_cancelExternalEncoderCommand
_cancelFollowingObjectCommand
_cancelPathObjectCommand
_changeEnableModeOfAdditionObjectIn
_changeEnableModeOfControllerObjectIn
_changeEnableModeOfFormulaObjectIn
_changeEnableOfFormula
_changeOperationMode
_checkEqualTask
_checkExistingUnitDataSet
_configurationManagement
_continue
_continue_V3_1
_continuePath
_continuePath_V4_2
_ControllerObjectType
_copyTControllerShadow
_cpuData
_cpuDataRW
_DATE
_DATE_AND_TIME
_deactivateDpSlave
_deactivateTo
_deallocateTokenToVariableName

_defineFormula
_deleteAllUnitDataSets
_deleteUnitDataSet
_DINT
_disableAdditionObjectIn
_disableAxis
_disableAxis_V2_0
_disableAxis_V3_1
_disableAxisAdditiveTorque
_disableAxisInterface
_disableAxisSimulation
_disableAxisTorqueLimitNegative
_disableAxisTorqueLimitPositive
_disableCamming
_disableCamming_V3_0
_disableCamTrack
_disableCamTrackSimulation
_disableCommandToActual
_disableControllerObject
_disableControllerObjectIn
_disableExternalEncoder
_disableExternalEncoderSimulation
_disableFixedGearing
_disableFixedGearMotionIn
_disableFollowingObjectSimulation
_disableFollowingObjectSimulation_V3_1
_disableForceLimiting
_disableFormula
_disableFormulaObjectIn
_disableGearing
_disableGearing_V3_0
_disableMaster_V3_0
_disableMeasuringInput
_disableMeasuringInputSimulation
_disableMonitoringOfEncoderDifference

_disableMovingToEndStop
_disableOutputCam
_disableOutputCamSimulation
_disablePathObjectSimulation
_disableQFAxis
_disableQFAxis_V3_0
_disableQFAxis_V3_1
_disableScheduler
_disableSensor
_disableTorqueLimiting
_disableVelocityGearing
_disableVelocityLimiting
_disableVelocityLimiting_V4_3
_driveStates
_DWORD_FROM_2WORD
_DWORD_FROM_4BYTE
_DWORD_TO_2WORD
_DWORD_TO_4BYTE
_enableAdditionObjectIn
_enableAxis
_enableAxis_V2_0
_enableAxis_V3_1
_enableAxis_V3_2
_enableAxisAdditiveTorque
_enableAxisInterface
_enableAxisSimulation
_enableAxisTorqueLimitNegative
_enableAxisTorqueLimitPositive
_enableCamming
_enableCamming_V3_0
_enableCamTrack
_enableCamTrackSimulation
_enableCommandToActual
_enableControllerObject
_enableControllerObjectIn

_enableDistributedMotionDelayValueCalculation
_enableDpInterfaceSynchronizationMode
_enableExternalEncoder
_enableExternalEncoderSimulation
_enableFixedGearing
_enableFixedGearMotionIn
_enableFollowingObjectSimulation
_enableFollowingObjectSimulation_V3_1
_enableForceControlByCondition
_enableForceControlByCondition_V2_1
_enableForceControlByCondition_V3_1
_enableForceControlByCondition_V4_0
_enableForceLimitingByCondition
_enableForceLimitingByCondition_V4_0
_enableForceLimitingValue
_enableForceLimitingValue_V3_1
_enableFormula
_enableFormulaObjectIn
_enableGearing
_enableGearing_V3_0
_enableGearing_V3_1
_enableMaster_V3_0
_enableMeasuringInput
_enableMeasuringInputCyclic
_enableMeasuringInputCyclic_V3_2
_enableMeasuringInputSimulation
_enableMonitoringOfEncoderDifference
_enableMotionInPositionLockedForceLimitingProfile
_enableMotionInPositionLockedVelocityLimitingProfile
_enableMotionInPositionLockedVelocityLimitingProfile_V4_0
_enableMotionInPositionLockedVelocityLimitingProfile_V4_3
_enableMovingToEndStop
_enableMovingToEndStop_V2_0
_enableOutputCam
_enableOutputCamSimulation

_enablePathObjectSimulation
_enablePathObjectTracking
_enablePathObjectTrackingSuperimposed
_enablePositionLockedForceLimitingProfile
_enablePositionLockedForceLimitingProfile_V3_1
_enablePositionLockedVelocityLimitingProfile
_enablePositionLockedVelocityLimitingProfile_V4_0
_enableQFAxis
_enableQFAxis_V3_0
_enableQFAxis_V4_0
_enableScheduler
_enableSensor
_enableTimeLockedForceLimitingProfile
_enableTimeLockedForceLimitingProfile_V3_1
_enableTimeLockedVelocityLimitingProfile
_enableTimeLockedVelocityLimitingProfile_V4_0
_enableTimeLockedVelocityLimitingProfile_V4_3
_enableTorqueLimiting
_enableTorqueLimiting_V2_0
_enableVelocityGearing
_enableVelocityLimitingValue
_enableVelocityLimitingValue_V4_0
_enableVelocityLimitingValue_V4_3
_exportUnitDataSet
_exportUnitDataSet_01
_exportUserDataSet
_FALSE
_FB_MF_MultiAxis
_FB_MF_SingleAxis
_FB_MF_STGP
_FB_STGP_BasicMC
_FB_STGP_Cam
_FB_STGP_Cam_Ext
_FB_STGP_Gear
_FB_STGP_Path

_FB_STGP_Position
_finite
_firstIndexOf
_FixedGearType
_forceTControllerIdentification
_FormulaObjectType
_getActiveDpSlaveAddress
_getActiveNameOfStation
_getActuatorErrorNumberState
_getActuatorErrorState
_getAdditionObjectErrorNumberState
_getAdditionObjectErrorState
_getAlarmId
_getAverageTaskIdRunTime
_getAverageTaskRunTime
_getAxisDataSetParameter
_getAxisDataSetParameter_V3_1
_getAxisDataSetParameter_V4_1
_getAxisErrorNumberState
_getAxisErrorState
_getAxisInternalPosition
_getAxisSpecificState
_getAxisSpecificState2
_getAxisStoppingData
_getAxisUserPosition
_getBit
_getCamErrorNumberState
_getCamErrorState
_getCamFollowingDerivative
_getCamFollowingMinMax
_getCamFollowingValue
_getCamLeadingValue
_getCamSpecificState
_getCamSpecificState2
_getCamTrackErrorNumberState

_getCamTrackErrorState
_getCamTrackSpecificState
_getCircularPathData
_getCircularPathData_V4_2
_getCircularPathGeometricData
_getCommandId
_getConfigurationData
_getControllerObjectErrorNumberState
_getControllerObjectErrorState
_getCurrentTaskIdRunTime
_getCurrentTaskRunTime
_getDataByToken
_getDeviceId
_getDoIndexNumberFromLogAddress
_getDpStationAddressFromLogDiagnosticAddress
_getExternalEncoderErrorNumberState
_getExternalEncoderErrorState
_getExternalEncoderSpecificState
_getExternalEncoderSpecificState2
_getFixedGearErrorNumberState
_getFixedGearErrorState
_getFollowingObjectErrorNumberState
_getFollowingObjectErrorState
_getForceControlDataSetParameter
_getForceControlDataSetParameter_V3_1
_getForceControlDataSetParameter_V4_1
_getFormulaObjectErrorNumberState
_getFormulaObjectErrorState
_getGearAxisSpecificState
_getGearAxisSpecificState2
_getGeoAddressFromLogAddress
_getInOutByte
_getInternalTaskIdIdx
_getInternalTaskIdx
_getInternalTimeStamp

_getIO_Part_4
_getIPConfig
_getLinearPathData
_getLinearPathData_V4_2
_getLinearPathGeometricData
_getLogDiagnosticAddressFromDpStationAddress
_getLogicalAdressOfIoVariable
_getMasterValue
_getMaximalTaskIdRunTime
_getMaximalTaskRunTime
_getMeasuringInputErrorNumberState
_getMeasuringInputErrorState
_getMeasuringInputSpecificState
_getMeasuringInputSpecificState2
_getMemoryCardId
_getMinimalTaskIdRunTime
_getMinimalTaskRunTime
_getMotionStateOfAxisCommand
_getMotionStateOfFollowingObjectCommand
_getMotionStateOfPathObjectCommand
_getNextLogAddress
_getOutputCamErrorNumberState
_getOutputCamErrorState
_getOutputCamSpecificState
_getOutputCamSpecificState2
_getPathAxesData
_getPathAxesPosition
_getPathCartesianData
_getPathCartesianPosition
_getPathGeometricData
_getPathObjectBcsFromOcsData
_getPathObjectErrorNumberState
_getPathObjectErrorState
_getPathObjectOcsFromBcsData
_getPendingAlarms

_getPnInterfacePortNeighbour
_getPNSyncCounter
_getPolynomialPathData
_getPolynomialPathData_V4_2
_getPolynomialPathGeometricData
_getPosAxisSpecificState
_getPosAxisSpecificState2
_getProgrammedTargetPosition
_getQFAxisDataSetParameter
_getQFAxisDataSetParameter_V3_1
_getSafeValue
_getSegmentIdentification
_getSensorErrorNumberState
_getSensorErrorState
_getSlaveValue
_getStateOfActuatorCommand
_getStateOfAdditionObjectCommand
_getStateOfAllDpSlaves
_getStateOfAllDpStations
_getStateOfAxisCommand
_getStateOfCamCommand
_getStateOfCamTrackCommand
_getStateOfControllerObjectCommand
_getStateOfDiagnosticDataCommand
_getStateOfDpSlave
_getStateOfExternalEncoderCommand
_getStateOfFixedGearCommand
_getStateOfFollowingObjectCommand
_getStateOfFormulaObjectCommand
_getStateOfIO
_getStateOfMeasuringInputCommand
_getStateOfMotionBuffer
_getStateOfOutputCamCommand
_getStateOfPathObjectCommand
_getStateOfPathObjectMotionBuffer

_getStateOfProcessInterruptCommand
_getStateOfRecordCommand
_getStateOfSensorCommand
_getStateOfSingleDpSlave
_getStateOfTask
_getStateOfTaskId
_getStateOfTo
_getStateOfUnitDataSetCommand
_GetStateOfXCommand
_getStationType
_getSyncCommandId
_getSystemTime
_getTaskId
_getTimeDifferenceOfInternalTimeStamp
_getTimeDifferenceOfInternalTimeStamps
_homing
_imData
_IMPLEMENTATION
_importUnitDataSet
_importUnitDataSet_01
_importUserDataSet
_INT
_INTERFACE
_INTERFACE_AND_IMPLEMENTATION
_internalTest
_interpolateCam
_isNaN
_lastIndexOf
_lengthIndexOf
_LINT
_loadUnitDataSet
_loadUnitDataSet_01
_LREAL
_MC_CamIn
_MC_CamIn_V4_0

_MC_CamInMode
_MC_CamOut
_MC_CamSwitch
_MC_CamSwitchDirection
_MC_CamSwitchOff
_MC_Direction
_MC_DisableMode
_MC_EnableMode
_MC_EncoderMode
_MC_GearIn
_MC_GearInMode
_MC_GearOut
_MC_Home
_MC_HomingMode
_MC_Jog
_MC_MoveAbsolute
_MC_MoveAdditive
_MC_MoveRelative
_MC_MoveSuperimposed
_MC_MoveVelocity
_MC_Phasing
_MC_PositionProfile
_MC_Power
_MC_Power_V4_0
_MC_ReadActualPosition
_MC_ReadAxisError
_MC_ReadBoolParameter
_MC_ReadBoolParameter_V4_0
_MC_ReadParameter
_MC_ReadParameter_V4_0
_MC_ReadStatus
_MC_Reset
_MC_Stop
_MC_StopMode
_MC_VelocityProfile

_MC_WriteBoolParameter
_MC_WriteBoolParameter_V4_0
_MC_WriteParameter
_MC_WriteParameter_V4_0
_modifyCountES
_modifyCountRT
_move
_movePathCircular
_movePathCircular_V4_1
_movePathCircular_V4_3
_movePathLinear
_movePathLinear_V4_1
_movePathLinear_V4_3
_movePathPolynomial
_movePathPolynomial_V4_1
_movePathPolynomial_V4_3
_movePointToPoint
_movePointToPoint_V4_1
_MRES
_NOT
_OR
_PathAxis
_PathObjectTpye
_pos
_readDiagnosticData
_readDriveFaults
_readDriveMultiParameter
_readDriveMultiParameterDescription
_readDriveParameter
_readDriveParameterDescription
_readRecord
_readSystemClock
_readVariableDiagnosticData
_readVariableRecord
_REAL

_redefineExternalEncoderPosition
_redefinePathObjectOcs
_redefinePosition
_releaseSemaphore
_removeBufferedActuatorCommandId
_removeBufferedAdditionObjectCommandId
_removeBufferedAxisCommandId
_removeBufferedCamCommandId
_removeBufferedCamTrackCommandId
_removeBufferedControllerObjectCommandId
_removeBufferedExternalEncoderCommandId
_removeBufferedFixedGearCommandId
_removeBufferedFollowingObjectCommandId
_removeBufferedFormulaObjectCommandId
_removeBufferedMeasuringInputCommandId
_removeBufferedOutputCamCommandId
_removeBufferedPathObjectCommandId
_removeBufferedSensorCommandId
_reset_AllAlarmId
_resetActuator
_resetActuatorConfigDataBuffer
_resetActuatorError
_resetAdditionObject
_resetAdditionObjectConfigDataBuffer
_resetAdditionObjectError
_resetAlarmId
_resetAllAlarmId
_resetAxis
_resetAxis_V2_0
_resetAxis_V3_0
_resetAxisConfigDataBuffer
_resetAxisError
_resetAxisError_V3_0
_resetCam
_resetCam_V2_0

_resetCam_V3_0
_resetCam_V4_2
_resetCamConfigDataBuffer
_resetCamError
_resetCamError_V3_0
_resetCamTrack
_resetCamTrackConfigDataBuffer
_resetCamTrackError
_resetControllerObject
_resetControllerObjectConfigDataBuffer
_resetControllerObjectError
_resetDriveObjectFault
_resetExternalEncoder
_resetExternalEncoder_V2_0
_resetExternalEncoder_V3_0
_resetExternalEncoderConfigDataBuffer
_resetExternalEncoderError
_resetExternalEncoderError_V3_0
_resetFixedGear
_resetFixedGearConfigDataBuffer
_resetFixedGearError
_resetFollowingObject
_resetFollowingObject_V2_0
_resetFollowingObject_V3_0
_resetFollowingObject_V3_1
_resetFollowingObjectConfigDataBuffer
_resetFollowingObjectError
_resetFollowingObjectError_V3_0
_resetFormulaObject
_resetFormulaObjectConfigDataBuffer
_resetFormulaObjectError
_resetMeasuringInput
_resetMeasuringInput_V2_0
_resetMeasuringInput_V3_0
_resetMeasuringInputConfigDataBuffer

_resetMeasuringInputError
_resetMeasuringInputError_V3_0
_resetMotionBuffer
_resetOutputCam
_resetOutputCam_V2_0
_resetOutputCam_V3_0
_resetOutputCamConfigDataBuffer
_resetOutputCamError
_resetOutputCamError_V3_0
_resetPathObject
_resetPathObjectConfigDataBuffer
_resetPathObjectError
_resetPathObjectMotionBuffer
_resetSensor
_resetSensorConfigDataBuffer
_resetSensorError
_resetTask
_resetTaskId
_resetTController
_resetTControllerError
_resetTechnologicalErrors
_resetUnitData
_restartTask
_restartTaskId
_resumeTask
_resumeTaskId
_RETAIN
_retriggerTaskControlTime
_retriggerTaskIdControlTime
_RUN
_runMotionInPositionLockedForceProfile
_runMotionInPositionLockedVelocityProfile
_runPositionBasedMotionIn
_runPositionBasedMotionIn_V3_2
_runPositionLockedForceProfile

_runPositionLockedVelocityProfile
_runTimeLockedForceProfile
_runTimeLockedForceProfile_V3_1
_runTimeLockedPositionProfile
_runTimeLockedVelocityProfile
_runVelocityBasedMotionIn
_runVelocityBasedMotionIn_V3_2
_S7_COUNTER
_S7_TIMER
_saveConfigData
_savePersistentMemoryData
_saveUnitDataSet
_saveUnitDataSet_01
_SC_ALARM_CONFIGURATION
_SC_ARRAY_BOUND_ERROR_READ
_SC_ARRAY_BOUND_ERROR_WRITE
_SC_BACKGROUND_TIMER_OVERFLOW
_SC_BATTERY_WARNING_LEVEL1
_SC_BATTERY_WARNING_LEVEL2
_SC_CYCLE_TIMER_OVERFLOW
_SC_DEVICE_COMMAND
_SC_DIAGNOSTIC_INTERRUPT
_SC_DIVISION_BY_ZERO
_SC_DP_CLOCK_DETECTED
_SC_DP_SLAVE_NOT_SYNCHRONIZED
_SC_DP_SLAVE_SYNCHRONIZED
_SC_DP_SYNCHRONIZATION_LOST
_SC_DRIVE_OBJECT_ALARM
_SC_DRIVE_OBJECT_FAULT
_SC_EXCEPTION
_SC_EXTERNAL_COMMAND
_SC_IMAGE_UPDATE_FAILED
_SC_IMAGE_UPDATE_OK
_SC_INVALID_ADDRESS
_SC_INVALID_FLOATING_POINT_OPERATION

_SC_IO_MODULE_NOT_SYNCHRONIZED
_SC_IO_MODULE_SYNCHRONIZED
_SC_MODE_SELECTOR
_SC_PC_INTERNAL_FAILURE
_SC_PROCESS_INTERRUPT
_SC_PULL_PLUG_INTERRUPT
_SC_STATION_DISCONNECTED
_SC_STATION_RECONNECTED
_SC_TO_INSTANCE_NOT_EXISTENT
_SC_VARIABLE_ACCESS_ERROR_READ
_SC_VARIABLE_ACCESS_ERROR_WRITE
_sendProcessInterrupt
_SensorType
_SERVICE
_setAndGetEncoderValue
_setAxisDataSetActive
_setAxisDataSetParameter
_setAxisDataSetParameter_V3_1
_setAxisDataSetParameter_V4_1
_setAxisSTW
_setAxisSTW_V4_0
_setBit
_setCammingOffset
_setCammingOffset_V1_1
_setCammingOffset_V3_1
_setCammingScale
_setCammingScale_V1_1
_setCammingScale_V3_1
_setCamOffset
_setCamScale
_setCamTrackState
_setControllerObjectPIDControl
_setDataByToken
_setDeviceErrorLED
_setDpSlaveAddress

_setDriveObjectSTW
_setExternalEncoderValue
_setFixedGearingOffset
_setFixedGearMaster
_setForceCommandValue
_setForceCommandValue_V3_1
_setForceControlDataSetParameter
_setForceControlDataSetParameter_V3_1
_setForceControlDataSetParameter_V4_1
_setFormula
_setFormulaObjectOutputValue
_setGearingOffset
_setGearingOffset_V3_1
_setIO_Part_4
_setIPConfig
_setIpoClockBorderToPosition
_setMaster
_setMaster_V3_0
_setMasterValue_V3_0
_setModeSelfAdaptingConfiguration
_setNameOfStation
_setOutputCamCounter
_setOutputCamState
_setOutputCamState_V4_0
_setPathObjectOcs
_setQFAxisDataSetParameter
_setQFAxisDataSetParameter_V3_1
_setQFAxisFCharacteristics
_setQFAxisQCharacteristics
_setSafeValue
_setSystemTime
_setTControllerActualIdentificationType
_setTControllerControlRangeParameter
_setTControllerCycleParameter
_setTControllerCycleParameterSecondary

_setTControllerDPIDParameter
_setTControllerDPIDParameterSecondary
_setTControllerIdentificationModifiedTangentMethodParameter
_setTControllerIdentificationModifiedTangentMethodProcessParameter
_setTControllerIdentificationStandardTangentMethodParameter
_setTControllerIdentificationStandardTangentMethodProcessParameter
_setTControllerInputDisplayValueParameter
_setTControllerInputFilterParameter
_setTControllerInputGradientCheckParameter
_setTControllerInputLimitCheckParameter
_setTControllerIntegrator
_setTControllerIntegratorSecondary
_setTControllerLowerPlausibilityParameter
_setTControllerLowerPlausibilityParameterSecondary
_setTControllerManualOutputValue
_setTControllerOperatingMode
_setTControllerProcessModeParameter
_setTControllerPWMParameter
_setTControllerPWMParameterSecondary
_setTControllerSetpoint
_setTControllerUpperPlausibilityParameter
_setTControllerUpperPlausibilityParameterSecondary
_SHUTDOWN
_SINT
_sizeOf
_startSyncCommands
_startTask
_startTaskId
_STARTUP
_startupData
_stop
_stop_V3_0
_stop_V4_2
_stopEmergency
_stopEmergency_V3_0

_stopEmergency_V4_2
_stopPath
_stopPath_V4_1
_STOPU
_StructCamTrackArrayOfSingleCamSettings
_StructCamTrackSingleCamSettings
_suspendTask
_suspendTaskDebug
_suspendTaskId
_suspendTaskIdDebug
_synchronizeDpInterfaces
_synchronizeExternalEncoder
_taskTraceStart
_taskTraceStop
_taskTraceUserEvent
_taskTraceWriteout
_tcpCloseConnection
_tcpCloseServer
_tcpOpenClient
_tcpOpenServer
_tcpReceive
_tcpSend
_testAndSetSemaphore
_testSFBSysDataInit
_TIME
_TIME_OF_DAY
_toggleBit
_triggerCommissioningMode
_triggerTestMode
_TRUE
_UDINT
_udpAddMulticastGroupMembership
_udpDropMulticastGroupMembership
_udpReceive
_udpSend

_UINT
_ULINT
_upsData
_USINT
_waitTime
_WORD_FROM_2BYTE
_WORD_TO_2BYTE
_writeAndSendMessage
_writeDriveMultiParameter
_writeDriveParameter
_writeRecord
_writeVariableRecord
_XOR
_Xreceive
_Xsend

A

ABORT_CONNECTION
ABORT_CURRENT_COMMAND
ABORTED
abortPosition
ABS
ABSOLUTE
absoluteEncoder
ACCELERATING
ACCELERATING_FORCE
ACCESS_DENIED
accessState
ACCORDING_TO_DEFINITION_ORDER
ACOS
ACTION
ACTIVATE_CHANGED_CONFIG_DATA
ACTIVATE_CONFIGURATION_DATA
ACTIVATE_FALL_BACK_CONFIGURATION
ACTIVATE_RESTART

ACTIVATED
ACTIVATED_NO_ALARM
activationModeChangedConfigData
ACTIVE
ACTIVE_ABSOLUTE
ACTIVE_AND_WAITING_FOR_CAM_TRACK_OUTPUT_INACTIVE
ACTIVE_AND_WAITING_FOR_DATA
ACTIVE_AND_WAITING_FOR_NEXT_CYCLE
ACTIVE_HOMING
ACTIVE_RELATIVE
ACTIVE_WITH_CIRCULAR_MOVE
ACTIVE_WITH_CONSTANT_LIMITS
ACTIVE_WITH_DYNAMIC_ADAPTION
ACTIVE_WITH_POLYNOMIAL_MOVE
ACTIVE_WITH_VARIABLE_LIMITS
ACTIVE_WITHOUT_DYNAMIC_ADAPTION
activeCam
activeMaster
ACTOR_ADAPTION_DATA
actorData
actorMonitoring
ACTUAL
ACTUAL_ACTIVATED
ACTUAL_AND_DEFAULT_VALUE
ACTUAL_DIRECTION_PASSIVE
ACTUAL_VALUE
ACTUAL_VALUE_INSIDE_POSITIONING_WINDOW
ACTUAL_VALUE_OUT_OF_POSITIONING_WINDOW
actualCycleData
ActualDPIDData
actualIdentificationModifiedTangentMethodData
actualIdentificationStandardTangentMethodData
actualIdentificationType
actualInputData
actualInputLimitCheckData

actualInputState
actualTorque
actualValueDefault
actualValueIn
ADAPTED_AND_DIFFERENT
ADAPTED_AND_EQUAL
ADAPTION_ERROR
ADD
ADD_DT_TIME
ADD_TIME
ADD_TOD_TIME
additionalSensorData
additionResult
ADDITIVE
additiveTorque
additiveTorqueIn
ADVANCED_TYPE
ALARM_S
ALARM_SQ
ALARMS_ERROR
ALARMS_QSTATE
ALARMS_STATE
ALL
ALL_ADAPTION_DATA
ALL_AXIS_MOTION
ALL_EDGES
ALL_ERRORS
ALL_GLOBAL
ALL_ID
ALL_POSITION_RELATED_MOTION
ANALOG
ANALOG_TEMPERATURE
AND
ANDN
ANYOBJECT

ANYOBJECT_TO_OBJECT
AnyType_to_BigByteArray
ANYTYPE_TO_LITTLEBYTEARRAY
APPLICATION
APPROACH_MOVE
APPROACH_NEGATIVE_PASSIVE
APPROACH_POSITIVE_PASSIVE
ARRAY
AS
ASIN
ASSEMBLY_BASE_DRIVE
ASSEMBLY_BASE_EXTERN
ASSEMBLY_BASE_LINEAR
ASSEMBLY_BASE_LOAD
AT
AT_DECELERATION_START
AT_END_OF_COMMAND
AT_MOTION_START
AT_PROFILE_START
AT_SYNCHRONIZING_START
AT_THE_END_OF_CAM_CYCLE
ATAN
ATTACHED_STEADILY
AUTOMATIC_INTERFACE_SYNCHRONIZATION
AUTOMATICALLY
AVAILABLE
AVERAGING
AXIS_HOMED
AXIS_STOPPED_AT_POSITION

B

B_SPLINE
BASIC
BASIC_AND_SUPERIMPOSED_MOTION_ACTIVE
BASIC_AND_SUPERIMPOSED_POS_MOTION_ACTIVE

BASIC_MOTION
BASIC_MOTION_ACTIVE
BASIC_MOTION_ACTIVE_WITH_LENGTH_OUTPUT
BASIC_NON_POS_AND_SUPERIMPOSED_POS_MOTION_ACTIVE
BASIC_POS_AND_SUPERIMPOSED_NON_POS_MOTION_ACTIVE
BASIC_POS_MOTION_ACTIVE
basicMotion
Baudrate_1
Baudrate_2
Baudrate_3
Baudrate_4
Baudrate_5
BCD_TO_BYTE
BCD_TO_DINT
BCD_TO_DWORD
BCD_TO_INT
BCD_TO_LWORD
BCD_TO_SINT
BCD_TO_WORD
bcs
BE_SYNCHRONOUS_AT_POSITION
BEGIN_SYNC
BEGIN_TO_STOP_WHEN_POSITION_REACHED
BigByteArray_to_AnyType
BIGBYTEARRAY_TOANYTYPE
BIN_CODE
BOOL
BOOL_TO_BYTE
BOOL_TO_DWORD
BOOL_TO_WORD
BOOL_VALUE_TO_DINT
BOOL_VALUE_TO_INT
BOOL_VALUE_TO_LREAL
BOOL_VALUE_TO_REAL
BOOL_VALUE_TO_SINT

BOOL_VALUE_TO_UDINT
BOOL_VALUE_TO_UINT
BOOL_VALUE_TO_USINT
BOTH
BOTH_DIRECTION
BOTH_EDGES
BOTH_EDGES_FIRST_FALLING
BOTH_EDGES_FIRST_RISING
BUFFERED
BY
BY_CAM_TRACK_END
BY_CENTER_AND_ARC
BY_COMMAND
BY_END_POSITION
BY_PROFILE_END
BY_START_POSITION
BY_STW_BIT
BY_VALUE
BYTE
BYTE_TO_BCD
BYTE_TO_BOOL
BYTE_TO_DINT
BYTE_TO_DWORD
BYTE_TO_INT
BYTE_TO_SINT
BYTE_TO_UDINT
BYTE_TO_UINT
BYTE_TO_USINT
BYTE_TO_WORD
BYTE_VALUE_TO_LREAL
BYTE_VALUE_TO_REAL

C

C_SPLINE
CACHE_ID

CAL
CALC
CALCN
CAM_AND_ZM_PASSIVE
CAM_DATA_RESET
CAM_PASSIVE
CAMMING
cammingAdjustments
CAMTRACK_DISABLE
camTrackPosition
CamType
CASE
CHANGE_DS_IS_LOCKED
CHANGE_FAILED
CHANNEL_0
CHANNEL_1
CHANNEL_2
CHANNEL_3
CHANNEL_4
CHANNEL_5
CHANNEL_6
CHANNEL_7
CIRCULAR
circularPathCommand
CLAMP_BY_FOLLOWING_ERROR_DEVIATION
CLAMP_WHEN_TORQUE_LIMIT_REACHED
CLOSE_ON_EXIT
CMD_CONTINUEAXIS
CMD_DISABLEAXIS
CMD_DISABLEAXISSIMULATION
CMD_ENABLEAXIS
CMD_ENABLEAXISSIMULATION
CMD_ENABLEMEASURINGINPUT
CMD_HOMING
CMD_INIT

CMD_MOVE
CMD_POSABS
CMD_POSREL
CMD_RESETAXIS
CMD_SETDATASETPARAM
CMD_STOPAXIS
CMD_STOPEMERGENCY
COLLECT_CHANGED_CONFIG_DATA
COMMAND_DONE
COMMAND_FAILED
COMMAND_ID
COMMAND_NOT_FOUND
COMMAND_VALUE
COMMAND_VALUE_BASIC_MOTION
COMMAND_VALUE_SUPERIMPOSED_MOTION
CommandIdType
COMPLETE_RANGE
CONCAT
CONCAT_DATE_TOD
CONDITION_1
CONDITION_1_AND_CONDITION_2
CONDITION_1_OR_CONDITION_2
CONDITION_2
CONFIG_TO_SHADOW
CONFIGURATION
CONFIGURATION_DELETED
CONFIGURATION_ERROR
CONFIGURATION_ID_NOT_FOUND
CONFIGURATION_NOT_FOUND
CONFIGURATION_VERSION
CONFIGURED
CONFIGURED_OUTPUT_VALUE
CONNECTED
connection
CONSTANT

CONSTANT_FORCE
CONSTANT_MOVE
CONTINUOUS
control
CONTROL_STOP
controlDeviation
controllerOutput
controlRangeParameter
COS
counterCamData
counterMeasuredValue1
counterMeasuredValue2
CRITICAL
CTD
CTD_DINT
CTD_LINT
CTD_UDINT
CTD_ULINT
CTU
CTU_DINT
CTU_LINT
CTU_UDINT
CTU_ULINT
CTUD
CTUD_DINT
CTUD_LINT
CTUD_UDINT
CTUD_ULINT
CURRENT
CURRENT_MODE
currentMasterData
currentSlaveData
currentSyncPosition
cycleParameter
cycleParameterSecondary

CYCLIC
CYCLIC_ABSOLUTE
CYCLIC_RELATIVE
cyclicMeasuringEnableCommand

D

DATA_EXCHANGE_INACTIVE
DATA_INCOMPATIBLE
DATA_INCOMPLETE
DATA_MISMATCH
DATASET_ALREADY_EXISTS
DATASET_ID_NOT_VALID
DATASET_NOT_FOUND
dataSetMonitoring
DATE
DATE_AND_TIME
DATE_AND_TIME_TO_DATE
DATE_AND_TIME_TO_TIME_OF_DAY
dccAux_2Clock
dccAuxClock
DEACTIVATED
DEACTIVATED_NO_ALARM
DECELERATING
DECELERATING_FORCE
DECODE_STOP
DEFAULT_MODE
DEFAULT_PASSIVE
DEFAULT_UNIT
DEFAULT_VALUE
defaultAdditiveTorque
defaultMotionIn
defaultTorqueLimitNegative
defaultTorqueLimitPositive
definitionState
DELETE

DEPENDING_ON_OUTER_INPUT_LIMITCHECK_STATUS
derivedValue
DESYNCHRONIZING
DIFFERENTIATION
DINT
DINT_TO_BCD
DINT_TO_BYTE
DINT_TO_DWORD
DINT_TO_INT
DINT_TO_LREAL
DINT_TO_REAL
DINT_TO_SINT
DINT_TO_STRING
DINT_TO_UDINT
DINT_TO_UINT
DINT_TO_USINT
DINT_TO_WORD
DINT_VALUE_TO_BOOL
DINTIn1
DINTIn1Default
DINTIn2
DINTIn2Default
DINTIn3
DINTIn3Default
DINTIn4
DINTIn4Default
DINTOut1
DINTOut1Default
DINTOut2
DINTOut2Default
DINTOut3
DINTOut3Default
DINTOut4
DINTOut4Default
DIRECT

DIRECT_HOMING
DIRECT_HOMING_RELATIVE
DIRECT_OUTPUT
DIRECT_TO_SHADOW
DIRECT_VALUE
DISABLE
DISABLE_ALL
DISABLE_CONTROLLER
DISABLE_DRIVE_IMMEDIATELY
DISABLE_MEASURE_AND_AVARAGE_OUTPUT_VALUE
DISABLE_MEASURE_AND_MANUAL_OUTPUT_VALUE
DISABLE_MOTION
DISABLE_OUTPUT
disableCommand
DISCONTINUOUS
distributedMotion
DISTURBED
DIV
DIVTIME
DO
DO_NOT_CHANGE
DO_NOT_CLAMP
DO_NOT_CLOSE_ON_EXIT
DO_NOT_SET_DP_ALARM
DONE
DP_1
DP_2
DP_3
DP_CLOCk_DETECTED
DP_INTERFACES_SYNCHRONIZED
DP_SLAVE_NOT_SYNCHRONIZED
DP_SLAVE_SYNCHRONIZED
DP_TEL1_STANDARD
DP_TEL101_611U_VELCTRL_NO_ENCODER
DP_TEL102_611U_POSCTRL_1_ENCODER

DP_TEL103_611U_POSCTRL_2_ENCODER

DP_TEL105_611U_DSC_1_ENCODER

DP_TEL106_611U_DSC_2_ENCODER

DP_TEL2_STANDARD

DP_TEL3_STANDARD

DP_TEL4_STANDARD

DP_TEL5_STANDARD

DP_TEL6_STANDARD

DP_TEL81_STANDARD

DP_TEL83_STANDARD

DPIDParameter

DPIDParameterSecondary

DPMASTER

DRIVE

DRIVE_INTERFACE

DriveAxis

driveData

DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE

DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED

DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID

DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR

DSC_SVS_DEVICE_ALARMS_IV_CALL

DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL

DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP

DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED

DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED

DSC_SVS_DEVICE_ALARMS_NO_ENTRY

DT

DT_TO_DATE

DT_TO_TOD

DWORD

DWORD_TO_BCD

DWORD_TO_BOOL

DWORD_TO_BYTE

DWORD_TO_DINT

DWORD_TO_INT
DWORD_TO_REAL
DWORD_TO_SINT
DWORD_TO_UDINT
DWORD_TO_UINT
DWORD_TO_USINT
DWORD_TO_WORD
DWORD_VALUE_TO_LREAL
DWORD_VALUE_TO_REAL
dynamicsIn
dynamicsInState

E

EDGE_NEG_SIDE_NEG_PASSIVE
EDGE_NEG_SIDE_POS_PASSIVE
EDGE_POS_SIDE_NEG_PASSIVE
EDGE_POS_SIDE_POS_PASSIVE
EFFECTIVE
effectiveData
effectiveTaskRuntime
ELSE
ELSIF
EMPTY
EN
ENABLE
ENABLE_OFFSET_OF_ABSOLUTE_ENCODER
enableCommand
enableForceControlByConditionCommand
enableForceLimitingByConditionCommand
enableValidCam
ENCODER_ADAPTION_DATA
ENCODER_DISABLE
END_ACTION
END_CASE
END_CONFIGURATION

END_EXPRESSION
END_FOR
END_FUNCTION
END_FUNCTION_BLOCK
END_IF
END_IMPLEMENTATION
END_INTERFACE
END_LABEL
END_OF_INTERPOLATION
END_OF_MOTION_STOP
END_POINT
END_PROGRAM
END_REPEAT
END_RESOURCE
END_STEP
END_STRUCT
END_SYNC
END_TRANSITION
END_TYPE
END_VAR
END_WAITFORCONDITION
END_WHILE
ENDAT
ENO
Enum_STGP_CMD
Enum_STGP_STATE
ENUM_TO_DINT
EnumAbsBaudrate
EnumAbsMsgFormat
EnumAbsMsgLength
EnumAbsoluteRelative
EnumAbsState
EnumAcceleration
EnumAccessMode
EnumActivatedDeactivated

EnumActiveAbsRelInactive
EnumActiveInactive
EnumActiveInactiveNoChange
EnumActiveNoChange
EnumActiveWaitingForDataInactive
EnumActualDirect
EnumActualValueFormat
EnumAdditionExecution
EnumAdditionObjectErrorReaction
EnumAlarmIdState
EnumAlarmIdType
EnumApproachPositionType
EnumAxisAbortAcceleration
EnumAxisActuatorInterfaceAllocationConfig
EnumAxisAdditionalSensorType
EnumAxisApproachDirection
EnumAxisCompareMode
EnumAxisControllerOutput
EnumAxisControllerType
EnumAxisCyclicSetUpInForceLimiting
EnumAxisDataAdaption
EnumAxisDelayValueCalculationMode
EnumAxisDelayValueState
EnumAxisDioActorType
EnumAxisDriverMode
EnumAxisEnableDscSpline
EnumAxisEnableMovingMode
EnumAxisEncoderAssemblyType
EnumAxisEncoderIdentification
EnumAxisEncoderMode
EnumAxisEncoderSystem
EnumAxisEncoderType
EnumAxisEncoderValueType
EnumAxisErrorReaction
EnumAxisExtrapolatedVelocitySwitch

EnumAxisFilterMode
EnumAxisFineInterpolatorMode
EnumAxisFollowingErrorConfigMode
EnumAxisFollowingMotionState
EnumAxisForceCommand
EnumAxisForceDerivativeLimitingMode
EnumAxisForceExtrapolationType
EnumAxisForceLimitingCommandType
EnumAxisForceProfileModeConditionCommand
EnumAxisForceProfileProcessingState
EnumAxisForceState
EnumAxisForceValueConditionCommand
EnumAxisFrictionReference
EnumAxisHomingMode
EnumAxisHomingReverseCamType
EnumAxisHomingState
EnumAxisIdentification
EnumAxisInterfaceActivationConfig
EnumAxisIntervalCounterMode
EnumAxisKindOfDataAdaption
EnumAxisLimitingProfileProcessingState
EnumAxisMotionCommand
EnumAxisMotionInCommandState
EnumAxisMotionState
EnumAxisMotorType
EnumAxisOperatingMode
EnumAxisPassiveApproachDirection
EnumAxisPassiveHomingMode
EnumAxisPathMotionState
EnumAxisPathPosToleranceCommandValue
EnumAxisPosCommandSpecification
EnumAxisProfileProcessingState
EnumAxisProgrammedTargetPosition
EnumAxisPulsesEnabledEvaluation
EnumAxisReferenceCamType

EnumAxisReferenceMaxNominal
EnumAxisSensorInterfaceAllocationConfig
EnumAxisServoMonitoringPositioningState
EnumAxisSideInput
EnumAxisStateMachineControl
EnumAxisSwitchingCondition
EnumAxisSwitchingCondition_1
EnumAxisSwitchingCondition_2
EnumAxisSwLimitModeSpecificMonitoring
EnumAxisTelegramType
EnumAxisTorqueForceReductionGranularity
EnumAxisType
EnumAxisUpdateCycle
EnumAxisUserDefaultHighLow
EnumAxisValueReferenceType
EnumAxisVelocityLimitingCommandType
EnumAxisVelocityLimitingDirection
EnumAxisVelocityLimitingValueMode
EnumBackLashDiff
EnumBackLashType
EnumBalanceFilterMode
EnumBlendingMode
EnumCamBSplineInterpolation
EnumCamCSplineInterpolation
EnumCamData
EnumCamDataMode
EnumCamDefinitionState
EnumCamErrorReaction
EnumCamExtremumType
EnumCamFctNoSolFRCause
EnumCamFctNoSolLRCause
EnumCamFctWrongSpecLRCause
EnumCamFollowingRangeSpecificationMode
EnumCamInterpolationMode
EnumCamLeadingRangeStartPoint

EnumCammingActivationMode
EnumCammingDirection
EnumCammingMode
EnumCamMode
EnumCamModify
EnumCamPositionMode
EnumCamRange
EnumCamRangeSpecification
EnumCamReferenceBySlaveModeRelative
EnumCamRepresentation
EnumCamTrackActivationMode
EnumCamTrackErrorReaction
EnumCamTrackNextCommand
EnumCamTrackOutputType
EnumCamTrackPositionReference
EnumCamTrackSetState
EnumCamTrackStartMode
EnumCamTrackState
EnumCamTrackStopMode
EnumCamTrackTaskLevel
EnumCamTrackType
EnumCamTrackValue
EnumCamValueType
EnumChangeMode
EnumCommandIdState
EnumCommandValueQuantizationMode
EnumCompactControllerControllerType
EnumConfigurationInfoId
EnumContinueDynamicType
EnumContinueSpecification
EnumControllerObjectErrorBehaviorMode
EnumControllerObjectErrorReaction
EnumControllerObjectInputType
EnumControllerObjectISetMode
EnumControllerObjectPrecontrolAddupMode

EnumControllerObjectValueBehaviorMode
EnumConversionDataType
EnumDataDefault
EnumDataSetChangingState
EnumDataType
EnumDecodeSequentialMotionCommand
EnumDefaultValueDirect
EnumDeviceAbortReadWriteJobs
EnumDeviceCommandIdState
EnumDeviceConfigurationActivationState
EnumDeviceDataActivationState
EnumDeviceDataScope
EnumDeviceDpAlarmMode
EnumDeviceIdType
EnumDeviceKindOfData
EnumDeviceModeOfOperation
EnumDeviceStartupOperationMode
EnumDeviceStateOfDpSlave
EnumDeviceStorageType
EnumDeviceUnitDataSetCommand
EnumDirectEffectiveUserDefault
EnumDirection
EnumDirectionType
EnumDisableQFAxisOutputEnableMode
EnumDisableQFAxisOutputMode
EnumDoNotChangeDirect
EnumDpInterfaceSyncMode
EnumDpInterfaceSyncState
EnumDpSegmentId
EnumDpSlaveSyncState
EnumDriveFaultsType
EnumEffectiveUserDefault
EnumEnableAxisMode
EnumEnableDisable
EnumEncoderErrorReaction

EnumEncoderIdentification
EnumEndBehaviourOfProfile
EnumErrorReporting
EnumErrorReset
EnumEventClass
EnumExtendedValue
EnumExtendedValueType
EnumFanBattery
EnumFctGenStartStop
EnumFctGenState
EnumFixedGearDirectEffectiveUserDefault
EnumFixedGearDirection
EnumFixedGearDisableMode
EnumFixedGearEnableMode
EnumFixedGearErrorBehaviorMode
EnumFixedGearErrorReaction
EnumFixedGearGearingMode
EnumFixedGearGearingType
EnumFixedGearMergeModeDisableGearing
EnumFixedGearMergeModeEnableGearing
EnumFixedGearMotionOutBehaviorMode
EnumFixedGearNextCommandDisableGearing
EnumFixedGearNextCommandEnableGearing
EnumFollowingObjectDynamicMergeMode
EnumFollowingObjectDynamicReference
EnumFollowingObjectErrorReaction
EnumFollowingObjectMotionImpact
EnumFollowingObjectStateSetMasterCommand
EnumFollowingObjectSynchronizeWithLookAhead
EnumFollowingObjectSynchronizingDirection
EnumFollowingObjectSynchronizingState
EnumFollowingObjectSyncMode
EnumFollowingObjectVelocityMode
EnumFollowingState
EnumFooActMasterConnectFailMaster

EnumFooInActMasterConnectFailMaster
EnumForceControlByConditionCommandState
EnumForceController
EnumForceControllerFilterType
EnumForceControllerState
EnumForceControllerType
EnumForceControllerTypeOfSensorData
EnumForceDirection
EnumForceLimitingByConditionCommandState
EnumFormulaErrorReaction
EnumFormulaObjectErrorBehaviorMode
EnumFormulaObjectOutBehaviorMode
EnumGearingActivationMode
EnumGearingDirection
EnumGearingMode
EnumGearingPosToleranceCommandValue
EnumGearingType
EnumGetValue
EnumHomingMode
EnumInactiveNoChange
EnumIncomingOutgoing
EnumInOutDirection
EnumInSensorIdentification
EnumInSensorMode
EnumInSensorSystem
EnumInSensorType
EnumInSensorValueType
EnumInsertMode
EnumIntegratorMode
EnumInterfaceID
EnumInterfaceValueDefaultValue
EnumInterpolationState
EnumIoIdType
EnumJerk
EnumLastValidInterfaceValueDefaultValue

EnumLeadingRangeEndPoint
EnumLimitExceededOk
EnumLogicOperation
EnumMasterMode
EnumMeasuredEdge
EnumMeasuringInputAccess
EnumMeasuringInputCyclicMode
EnumMeasuringInputErrorReaction
EnumMeasuringInputInputType
EnumMeasuringInputTaskLevel
EnumMeasuringRangeMode
EnumMeasuringState
EnumMemoryCardIdType
EnumMergeMode
EnumMergeModeDisableCamming
EnumMergeModeDisableGearing
EnumMergeModeEnableCamming
EnumMergeModeEnableGearing
EnumMergeModeEnableMotionIn
EnumMergeModeForceTimeProfile
EnumMergeModeStop
EnumModeSelfAdaptingConfiguration
EnumMotionBaseType
EnumMotionBufferState
EnumMotionCommandIdState
EnumMountSwitch
EnumMoveTimeOut
EnumMovingMode
EnumMovingModeStopCommand
EnumNextCommand
EnumNextCommandCamming
EnumNextCommandCancelCommand
EnumNextCommandDataAdaption
EnumNextCommandDelayValueCalculation
EnumNextCommandDisableClamping

EnumNextCommandDisableForceLimiting
EnumNextCommandDisableLimiting
EnumNextCommandDisableTorqueLimiting
EnumNextCommandEnable
EnumNextCommandEnableAxisInterface
EnumNextCommandEnableClamping
EnumNextCommandEnableForceControl
EnumNextCommandEnableForceLimitingValue
EnumNextCommandEnableLimitingValue
EnumNextCommandEnableMeasuring
EnumNextCommandEnableMotionIn
EnumNextCommandEnableTorqueLimiting
EnumNextCommandEncoderSimulation
EnumNextCommandForceProfile
EnumNextCommandForceValue
EnumNextCommandGearing
EnumNextCommandHoming
EnumNextCommandIpoClockBorderToPosition
EnumNextCommandMode
EnumNextCommandPathMove
EnumNextCommandProfile
EnumNextCommandReset
EnumNextCommandScaling
EnumNextCommandSensor
EnumNextCommandSetMaster
EnumNextCommandSyncEncoder
EnumNextEnablePathObjectMotionTracking
EnumOffsetMode
EnumOkFaulted
EnumOnOff
EnumOperationMode
EnumOutputCamErrorReaction
EnumOutputCamInvert
EnumOutputCamOutputType
EnumOutputCamPosition

EnumOutputCamState
EnumOutputCamTaskLevel
EnumOutputCamType
EnumOutputCamValue
EnumPathBlendingMode
EnumPathCartesianKinematicsType
EnumPathCircularDirection
EnumPathCircularType
EnumPathDataSource
EnumPathDynamicAdaption
EnumPathDynamicsIn
EnumPathDynamicsInLengthMode
EnumPathErrorReaction
EnumPathIjkMode
EnumPathKinematicsConfig2D
EnumPathKinematicsType
EnumPathMergeMode
EnumPathMode
EnumPathMotionBufferState
EnumPathMotionCommand
EnumPathMotionCommandIdState
EnumPathMotionState
EnumPathObjectAttachPath
EnumPathObjectBlendingAcceleration
EnumPathObjectCsType
EnumPathObjectOcsSettingType
EnumPathObjectRedefineOcsMode
EnumPathObjectTrackingSynchronizingMode
EnumPathPlane
EnumPathPointType
EnumPathPolynomialMode
EnumPathPositionIndication
EnumPathStopMode
EnumPathSyncAxisMotionState
EnumPathTransitionState

EnumPathTransitionType
EnumPathTransitionVelocityMode
EnumPathVelocity
EnumPathVelocityProfile
EnumPathVelocityProfileProcessingState
EnumPathWDirection
EnumPathWDynamics
EnumPathWMode
EnumPersistentDataState
EnumPositioningMode
EnumPositiveNegative
EnumPosValue
EnumProfile
EnumProfileDynamicsLimiting
EnumPulsesEnabledActiveInactiveNotEvaluated
EnumQFAxisOutputEnableMode
EnumQFAxisOutputMode
EnumQFAxisOutputSetMode
EnumRange
EnumRecognitionMode
EnumRedefineMode
EnumRedefineSpecification
EnumRemoveMode
EnumReqActDeactGetStateMode
EnumReqSysFunctMode
EnumRestartAxisCondition
EnumRestartDeniedCause
EnumSaveState
EnumScaleSpecification
EnumScalingSpecification
EnumScopeOfIO
EnumSensorDataType
EnumSensorErrorReaction
EnumSensorState
EnumSensorValueBehaviorMode

EnumSetAndGetSafeValue

EnumSetNoSet

EnumSlaveMode

EnumStartStop

EnumStateOfDpSlave

EnumStateOfDpStation

EnumStateOfIO

EnumStateOfMaintenanceOfIO

EnumStateOfTo

EnumStopDriveMode

EnumStopMode

EnumStopSpecification

EnumSyncCommandState

EnumSynchronizingMode

EnumSyncModeCamming

EnumSyncModeGearing

EnumSyncOffModeCamming

EnumSyncOffModeGearing

EnumSyncOffPositionReference

EnumSyncPositionReference

EnumSyncProfileReference

EnumSystemDirect

EnumTControllerActivationModeChangedConfigData

EnumTControllerBinaryIObits

EnumTControllerCommandDestination

EnumTControllerControllerType

EnumTControllerCopyShadow

EnumTControllerDataResetMode

EnumTControllerErrorReaction

EnumTControllerErrorResetMode

EnumTControllerExecutionLevel

EnumTControllerIdentificationAvailableType

EnumTControllerIdentificationModifiedTangentMethodStage

EnumTControllerIdentificationStandardTangentMethodStage

EnumTControllerIdentificationTransitionMode

EnumTControllerIdentificationType
EnumTControllerInputGradientCheckMode
EnumTControllerInputLimitCheckMode
EnumTControllerInputLimitCheckState
EnumTControllerInputType
EnumTControllerIntegrationLimitState
EnumTControllerIntegrationMode
EnumTControllerNextCommand
EnumTControllerOperatingMode
EnumTControllerOutputLimitState
EnumTControllerOutputType
EnumTControllerPlausibilityCheckMode
EnumTControllerPlausibilityState
EnumTControllerProcessControlState
EnumTControllerProcessMode
EnumTControllerRestartActivation
EnumTControllerRestartActivationSetting
EnumTControllerRestartCondition
EnumTControllerSimulationMode
EnumTcpNextCommandMode
EnumToActivationModeSetConfigData
EnumToCommandExecution
EnumToExecutionLevel
EnumToInvalidSysvarAccess
EnumToRestartActivation
EnumToRestartActivationSetting
EnumTorqueLimitUnitType
EnumToSetStateOfTo
EnumTraceState
EnumTrackingState
EnumTransferSuperimposedPosition
EnumTrueFalse
EnumUdpCommunicationMode
EnumUpsBatteryState
EnumUpsState

EnumUserDefaultDirect
EnumUserDefaultYesNo
EnumValueSpecification
EnumValueType
EnumVelocity
EnumXCommandIdState
EnumXCommunicationMode
EnumXNextCommandEnable
EnumYesNo
EQ
error
errorGroup
errorReaction
EXECUTED
EXISTING
EXIT
EXP
EXPD
EXPRESSION
EXPT
EXTENDED_LOOK_AHEAD
ExternalEncoderType
extrapolationData
extrapolationValue

F

F_EDGE
F_TRIG
FAILED
FALLING_EDGE
FALLING_EDGES_ONLY
FALSE
fanbattery
FAST
FAULTED

FCTGEN_DISABLED
FCTGEN_ENABLED
FCTGEN_LIMIT_EXCEEDED
FCTGEN_PARAM_VALID
FCTGEN_RUNNING
FCTGEN_START
FCTGEN_START_FG1
FCTGEN_START_FG2
FCTGEN_STARTING
FCTGEN_STOP
FCTGEN_STOPPING
FCTGEN_WAIT_FG1
FCTGEN_WAIT_FG2
FEEDBACK
FEEDBACK_EMERGENCY_STOP
FILTER_AND_CONSTANT_LIMIT
FIND
FINISHED
FIXED_GEAR_DISABLE
fixedGearValue
FOLLOWING_OBJECT_DISABLE
FOLLOWING_RANGE
FollowingAxis
FollowingObjectType
followingRange
followingRangeSettings
FOR
FORCE_AND_INPUT
FORCE_AND_POSITION
FORCE_AND_TIME
FORCE_CONDITION
FORCE_CONTROLLED
FORCE_LIMITED
FORCE_OR_INPUT
FORCE_OR_POSITION

FORCE_OR_TIME
FORCE_POSITION
FORCE_TIME
FORCE_VALUE
forceActualValueSettings
forceControllerData
forceControllerMonitorings
forceControllerSettings
forceLimitingCommand
forceMotionInPositionProfileCommand
forcePositionProfileCommand
forceStateData
forceTimeProfileCommand
FROM
FROM_BACKUP
FROM_FILE
FROM_RAM
FULL
FUNCTION
FUNCTION_BLOCK
FUNCTION_IS_ACTIVATED
FUNCTION_IS_ACTIVE
FUNCTION_IS_WAITING_FOR_ACTIVATION

G

GE
GEARING
GEARING_WITH_FRACTION
GEARING_WITH_RATIO
gearingAdjustments
GLOBAL_EXTREMUM
GOTO
GOTO
GRAY_CODE
GREATER_EQUAL

GT

H

HEAT_AND_COOL_OUTPUT_ZERO
HEAT_OUTPUT_ZERO
HEATING
HIGH
HIGH_VELOCITY
HOLD_CONNECTION
HOME_POSITION_ENTRY_MOVE
homingCommand
HW_TYPE

I

IDENTIFICATION
identificationModifiedTangentMethodParameter
identificationStandardTangentMethodParameter
IE_01
IE_02
IE_1
IE_2
IF
IMMEDIATELY
IMMEDIATELY_AND_BE_SYNCHRONOUS_AT_MASTER_POSITION
IMMEDIATELY_AND_SLAVE_POSITION
IMMEDIATELY_BY_CAM_TRACK_OUTPUT_INACTIVE
IMMEDIATELY_BY_TIME_PROFILE
IMMEDIATLY
IMPLEMENTATION
IN
IN_ACCELERATION
IN_ACTIVATION
IN_ADAPTION
IN_CALCULATION
IN_CONSTANT_MOTION

IN_DEACTIVATION
IN_DECELERATION
IN_DEFINED_TIME
IN_EXECUTION
IN_INTERPOLATION
IN_INTERPOLATION_BUT_NOT_ON_PROFILE
IN_MOTION
IN_OPERATION
INACTIVE
INACTIVE_AND_WAITING_FOR_NEXT_CYCLE
INCOMING
INCOMING_ALARM
INCREMENT_COUNTER
INCREMENT_DURATION
INITIAL_STEP
INPUT
INPUT_CONDITION
inputDisplayValueParameter
inputFilterParameter
inputGradientCheckParameter
inputLimitCheckParameter
INSERT
INT
INT_TO_BCD
INT_TO_BYTE
INT_TO_DINT
INT_TO_DWORD
INT_TO_LREAL
INT_TO_REAL
INT_TO_SINT
INT_TO_TIME
INT_TO_UDINT
INT_TO_UINT
INT_TO_USINT
INT_TO_WORD

INT_VALUE_TO_BOOL
INTERFACE
INTERFACE_VALUE
INTERNAL_ERROR
INTERNAL_RELATED
internalServoSettings
internalToTrace
INTERNEL_ERROR
INTERPOLATED
interpolation
INTERPOLATION_DONE
INTERRUPTED_COMMAND
INTERVAL_COUNTER
INVALID
INVALID_VALUE
IPO
IPO_2
IPO_FAST
ipoClock
ipoClock_2
ipoClock_fast

J

JMP
JMPC
JMPCN

K

kinematicsData

L

LABEL
LAST_INTERFACE_VALUE
LAST_OPERATION_MODE
LAST_VALID_INTERFACE_VALUE

LAST_VALUE
LD
LDN
LE
LEADING_RANGE
LEADING_RANGE_END
LEADING_RANGE_START
LEADING_RANGE_VALUE
leadingRange
leadingRangeSettings
LEFT
LEN
LENGTH_13
LENGTH_21
LENGTH_25
LESS
lifeSign
LIMIT
LIMIT_EXCEEDED
LIMITING_BY_DIRECT_VALUE
LIMITING_BY_USER_DEFAULT_VALUE
limitsOfPathDynamics
LINEAR
LINEAR_CONVERSIONDATA
LINEAR_SYSTEM
linearPathCommand
LINT
LittleByteArray_to_AnyType
LITTLEBYTEARRAY_TOANYTYPE
LN
LOAD_CONFIGURED_DATA
LOCAL_EXTREMUM
LOG
LONG_RUN_NEGATIVE
LONG_RUN_POSITIVE

LOW
LOW_VELOCITY
LOWER_PRIMARY_FAILED
LOWER_SECONDARY_FAILED
lowerPlausibilityParameter
lowerPlausibilityParameterSecondary
LREAL
LREAL_TO_DINT
LREAL_TO_INT
LREAL_TO_REAL
LREAL_TO_SINT
LREAL_TO_STRING
LREAL_TO_UDINT
LREAL_TO_UINT
LREAL_TO_USINT
LREAL_VALUE_TO_BOOL
LREAL_VALUE_TO_BYTE
LREAL_VALUE_TO_DWORD
LREAL_VALUE_TO_WORD
LREALIn1
LREALIn1Default
LREALIn2
LREALIn2Default
LREALIn3
LREALIn3Default
LREALIn4
LREALIn4Default
LREALOut1
LREALOut1Default
LREALOut2
LREALOut2Default
LREALOut3
LREALOut3Default
LREALOut4
LREALOut4Default

LT
LWORD
LWORD_TO_BCD

M

MAINTAIN_PROGRAMMED_DATA
MAINTENANCE_DEMANDED
MAINTENANCE_NO_INFO
MAINTENANCE_REQUIRED
MANDATORY
manualOutputValue
MASTER_RANGE
MASTER_SLAVE_ALARMMESSAGES_1
masterState
matchPosition
MAX
MAX_DYNAMICS
mcs
measuredValue1
measuredValue2
MEASURING_AND_MANUAL_OUTPUT
MEASURING_AND_OUTPUT_ZERO
MEASURING_ERROR
MEASURING_INPUT_DISABLE
MeasuringInputType
MEMORY_CARD_FULL
MEMORY_CARD_SERIAL_NUMBER
MID
MIN
MIN_MAX
minusLimitsOfDynamics
MOD
MODE_1
MODE_2
modeOfDpInterfaceSynchronization

modeOfOperation
MODIFICATION_ACTIVE
MODIFIED_TANGENTMETHOD
modulo
monitorings
MOTION_DONE
MOTION_EMERGENCY_ABORT
MOTION_EMERGENCY_STOP
MOTION_IN_POSITION_LOCKED_PROFILE
MOTION_STOP
motionBuffer
motionIn
MotionIn1
MotionIn1Default
MotionIn2
MotionIn2Default
MotionIn3
MotionIn3Default
MotionIn4
MotionIn4Default
MotionInDefault
MotionOut
MotionOut1
MotionOut1Default
MotionOut2
MotionOut2Default
MotionOut3
MotionOut3Default
MotionOutDefault
motionState
motionStateData
motionType
moveCommand
MOVING_WITH_REDUCED_VELOCITY
movingToEndStopCommand

MUL
MULTIME
MUX

N

NE
NEGATIVE
NEGATIVE_DIRECTION
NEXT_CAM_CYCLE
NEXT_CAM_TRACK_CYCLE
NEXT_MOTION
NEXT_WITH_REFERENCE
NO
NO_ACCESS
NO_ALARMMESSAGES
NO_CHANGE
NO_COMMAND_BUFFER_AVAILABLE
NO_CONSTRAINTS
NO_CYCLIC
NO_DP_SEGMENT
NO_HOMING_MODE
NO_POS_MOTION_ACTIVE
NO_REQUEST_TO_ACTIVATE
NO_RESSOURCES_AVAILABLE
NO_RESTART_ACTIVATION
NO_RETAIN_GLOBAL
NO_SET
NO_STORAGE_AVAILABLE
NO_TELEGRAM
NO_TYPE
NO_VALID_CONFIGURATION_ID
NO_VALID_STATE
NOCYCLIC
NODEF
NON_AVAILABLE

NONE
NOT
NOT
NOT_ACTIVATED
NOT_ADAPTED
NOT_APPLICABLE
NOT_EMPTY
NOT_ENOUGH_RAM
NOT_EVALUATED
NOT_EXISTENT
NOT_EXISTING
NOT_INTERPOLATED
NOT_LIMITED
NOT_MANDATORY
NOT_POSSIBLE
NOT_PRESENT
NOT_VALID
NOTSUPP
numberOfSummarizedTaskOverflow

O

O_K_
OBJECT
OCS
OF
OFF
OFFSET_MOVE
OFFSET_SCALE
OK
ON
ON_MASTER_AND_SLAVE_POSITION
ON_MASTER_POSITION
ON_POSITION
ON_PROFILE
ON_SLAVE_POSITION

ONBOARD
ONLY_POSITION_COMMAND
OPEN_POSITION_CONTROL
operatingMode
OPTIONAL
OPTIONAL_RTC
OR
ORDER_ID
ORN
OUT
OUTGOING
OUTGOING_ALARM
output
OUTPUT_CAM_DISABLE
OUTPUT_PATH_LENGTH
OUTPUT_PATH_LENGTH_ADDITIVE
OutputCamType
outputData
outputDefault
outputDerivative
outputDerivativeDefault
outputMonitoring
outputSettings
outputValue
outputValueSecondary
OVER
OVER_POSITION_TO_ENDPOSITION

P

override
PARABOLIC
PASSIVE_HOMING
path
pathMotion
pathSyncMotion

PD
PERMANENT_STORAGE
persistentDataPowerMonitoring
PID
PID_ACTUAL
pidController
pidControllerDefault
pidControllerEffective
pidControllerMonitorings
PINETREE
plusLimitsOfDynamics
PN_1
PN_2
POLYNOMIAL
polynomialPathCommand
PosAxis
posCommand
POSITION
POSITION_AND_DIRECT_NIST_B
POSITION_AND_INPUT
POSITION_AND_PROFIDRIVE_ENCODER_NIST_B
POSITION_AND_PROFIDRIVE_NIST_B
POSITION_AND_TIME
POSITION_CONDITION
POSITION_CONTROLLED
POSITION_LOCKED_PROFILE
POSITION_OR_INPUT
POSITION_OR_TIME
positionBasedMotionInCommand
positioningState
positionTimeProfileCommand
POSITIVE
POSITIVE_DIRECTION
POWER
precontrol

precontrolValueDefault
precontrolValueIn
PREDEFINED_APPLICATION_EVENTS
PREDEFINED_MODULE_INFORMATION
PREVIOUS_ACTIVATED
PRIMARY
processedValue
processModeParameter
PROFIDRIVE
PROFIDRIVE_NIST_A
PROFIDRIVE_NIST_B
PROFILE_DONE
PROFILE_END
PROGRAM
PROGRAM
PT1
PT2
PV
PWM
PWMParameter
PWMParameterSecondary

R

R_EDGE
R_TRIG
RAM_DISK_FULL
RANGE_EXTREMUM
rawValue
READ_AND_WRITE_JOBS
READ_ERROR
READ_JOBS
REAL
REAL_AXIS
REAL_TO_DINT
REAL_TO_DWORD

REAL_TO_INT
REAL_TO_LREAL
REAL_TO_SINT
REAL_TO_STRING
REAL_TO_TIME
REAL_TO_UDINT
REAL_TO_UINT
REAL_TO_USINT
REAL_VALUE_TO_BOOL
REAL_VALUE_TO_BYTE
REAL_VALUE_TO_DWORD
REAL_VALUE_TO_WORD
RECTANGLE_TTL
REDUNDANT
reference
REFERENCE_CAM_AND_EXTERNAL_ZM_PASSIVE
RELATE_SYNC_PROFILE_TO_LEADING_VALUE
RELATE_SYNC_PROFILE_TO_TIME
RELATIVE
RELEASE_DISABLE
REPEAT
REPLACE
REQUEST_ABORT
REQUEST_FALSE
REQUEST_TRUE
reset
RESOLVER
RESOURCE
RESTART_BAD_CONFIG
RESTART_BY_COMMAND
RESTART_BY_SYSVAR_AND_COMMAND
RESTART_COND_MISS
RESTART_NONE
RESTART_NOT_LAST_CONFIG
RESTART_NOT_READY

RESTART_SYSVAR_FAILED
restartActivation
RESULTING
RET
RETAIN
RETC
RETCN
RETURN
REVERSE
RIGHT
RIGHT_MARGIN
RISING_EDGE
RISING_EDGES_ONLY
ROL
ROR
ROTATORY
ROTATORY_SYSTEM
RS
RTC
RUN

S

SAFETY
SAME_DIRECTION
SAVE_ABORTED
SAVE_FINISHED
SEARCH_ENDPOINT
SEARCH_INFLECTIONPOINT
SEARCH_STARTPOINT
SECONDARY
SEL
SEMA
SENSOR_ABSOLUTE
SENSOR_ANALOG
SENSOR_CYCLIC_ABSOLUTE

SENSOR_INCREMENTAL
SENSOR_POSITION_DIFFERENCE_MEASUREMENT
sensorData
sensorMonitoring
sensorSettings
SEQUENTIAL
SERIAL_NUMBER
SERVO
SERVO_FAST
servoControlClock
servoControlClock_fast
servodata
servoMonitoring
servoSettings
servoTaskCycle
servoTaskCycle_fast
SET
SET_ACTUAL_VALUE
SET_DP_ALARM
SET_OFFSET_OF_ABSOLUTE_ENCODER_BY_POSITION
SET_VALUE
setForceCommandValueCommand
setMasterCommand
setpoint
SETPOINT_VALUE
setpointDefault
setpointIn
SETTING_OF_COEFFICIENTS
SETTING_UP
SHADOW
SHADOW_TO_DIRECT
SHL
SHORTEST_WAY
SHR
SIDE_INPUT_NEG

SIDE_INPUT_NO_SIDE
SIDE_INPUT_POS
SIMULATION
SIMULATION_ABORT
SIMULATION_STOP
SIN
SINGLE
SINGLE_PATH
singleCamState
SINT
SINT_TO_BCD
SINT_TO_BYTE
SINT_TO_DINT
SINT_TO_DWORD
SINT_TO_INT
SINT_TO_LREAL
SINT_TO_REAL
SINT_TO_UDINT
SINT_TO_UINT
SINT_TO_USINT
SINT_TO_WORD
SINT_VALUE_TO_BOOL
SINUS_1VPP
SINUSOIDAL
SLAVE_ALARMMESSAGES_1
SLAVE_RANGE
SLOW
SMOOTH
SPECIFIC
SPECIFIC_AXIS_MOTION
SPECIFIC_ERROR
SPECIFIC_ID
SPECIFIC_NUMBER
SPECIFIC_PART_OF_RANGE
SPECIFIC_POINT

SPECIFIC_START_DATA
specificVelocityProfile
SPEED_CONTROLLED
speedMode
SQRT
SR
SSI_MODE
ST
STANDARD
STANDARD_TANGENTMETHOD
STANDARD_TYPE
STANDSTILL
STANDSTILL_MONITORING_ACTIVE
START
START_POINT
STARTPOINT_ENDPOINT
state
STATE_HOMED
STATE_HOMING
STATE_INITIAL
STATE_MACHINE_CONTROL_BY_APPLICATION
STATE_MOVING
STATE_POSABS
STATE_POSITION_END
STATE_POSREL
STATE_STOPPED
STATE_TOACTIVE
stateCammingAdjustments
stateGearingAdjustments
stateOfDpInterfaceSynchronization
stateOfDpSlaveSynchronization
stateSetMasterCommand
STEP
STEPMOTOR
STN

STOP
STOP_ALL_FORMULA
STOP_AND_ABORT
STOP_AND_ABORT_AND_HOLD
STOP_DEVICE
STOP_IN_DEFINED_TIME
STOP_SPECIFIC_FORMULA
STOP_SYMMETRIC_WITH_POSITION
STOP_WITH_COMMAND_VALUE_ZERO
STOP_WITH_DYNAMIC_PARAMETER
STOP_WITH_MAXIMAL_DECELERATION
STOP_WITHOUT_ABORT
stopEmergencyCommand
STOPPED
STOPPED_WITHOUT_ABORT
STOPU
STRING
STRING_TO_DINT
STRING_TO_LREAL
STRING_TO_REAL
STRING_TO_UDINT
STRUCT
StructAdditionalSensorNumber
StructAdditionObjectMotionIn
StructAdditionObjectMotionOut
StructAlarmId
StructAlarmId_TO_DINT
StructAxisAbsoluteEncoder
StructAxisActorData
StructAxisActorMonitoring
StructAxisActualTorque
StructAxisAdditionalSensorData
StructAxisAdditiveTorque
StructAxisAdditiveTorqueIn
StructAxisDataSetReadWrite

StructAxisDataSetReadWrite_V3_1
StructAxisDataSetReadWrite_V4_1
StructAxisDefaultType
StructAxisDistributedMotion
StructAxisDriveData
StructAxisDriveSafetyExtendedFunctionsInfoData
StructAxisDynamicLimit
StructAxisDynamicQFData
StructAxisExtrapolationData
StructAxisForceActualValueSettings
StructAxisForceControlByConditionCommand
StructAxisForceControlDataSet
StructAxisForceControlDataSet_V3_1
StructAxisForceControlDataSet_V4_1
StructAxisForceControllerData
StructAxisForceControllerMonitorings
StructAxisForceControllerSettings
StructAxisForceLimitingByConditionCommand
StructAxisForceLimitingCommand
StructAxisForceMotionInPositionProfileCommand
StructAxisForcePositionProfileCommand
StructAxisForceStateData
StructAxisForceTimeProfileCommand
StructAxisHomingCommand
StructAxisHomingDefault
StructAxisInvertQOutput
StructAxisInvertSetPointHydraulicType
StructAxisModuloData
StructAxisMotionData
StructAxisMotionInData
StructAxisMotionStateData
StructAxisMoveCommand
StructAxisMovingToEndStopCommand
StructAxisOperatingData
StructAxisOverride

StructAxisPathMotion

StructAxisPathSyncMotion

StructAxisPosCommand

StructAxisPositionBasedMotionInCommand

StructAxisPositioningDefault

StructAxisPositioningState

StructAxisPositionTimeProfileCommand

StructAxisSensorData

StructAxisSensorMonitoring

StructAxisSensorSettings

StructAxisServoData

StructAxisServoMonitoring

StructAxisServoSettings

StructAxisSetForceCommandValueCommand

StructAxisSwLimit

StructAxisSwLimitState

StructAxisTorqueLimitIn

StructAxisTorqueLimitingCommand

StructAxisUserDefaultClamping

StructAxisUserDefaultForceLimiting

StructAxisUserDefaultTorqueLimit

StructAxisVelocityBasedMotionInCommand

StructAxisVelocityLimitingCommand

StructAxisVelocityMotionInPositionProfileCommand

StructAxisVelocityPositionProfileCommand

StructAxisVelocityTimeProfileCommand

StructCamFollowingRange

StructCamInterpolation

StructCammingAdjustments

StructCammingSettings

StructCamRange

StructCamScaleRange

StructCamSettings

StructCamTrackArrayOfSingleCamSettings

StructCamTrackEffectiveData

StructCamTrackSingleCamSettings
StructCamTrackUserDefault
StructCamUserDefault
StructClampingMonitoring
StructControllerDynamic
StructControllerObjectControlDeviation
StructControllerObjectOutputData
StructControllerObjectOutputMonitoring
StructControllerObjectPControllerData
StructControllerObjectPControllerDefault
StructControllerObjectPIDControllerData
StructControllerObjectPIDControllerDefault
StructControllerObjectPIDControllerEffective
StructControllerObjectPIDControllerMonitorings
StructControllerObjectPrecontrolData
StructControllerObjectValueIn
StructControllerType
StructControllerType_V3_1
StructCpuDriveStates
StructCyclicMeasuringEnableCommand
StructDataSetMonitoring
StructDeviceConfigurationData
StructDeviceConfigurationManagement
StructDeviceCpuData
StructDeviceCpuDataRW
StructDeviceDataActivationState
StructDeviceFanBattery
StructDeviceIm0
StructDeviceIm0Softwareversion
StructDeviceIm0Version
StructDeviceImData
StructDeviceStartUp
StructDeviceUpsData
StructDpStationAddressType
StructDynamicComp

StructDynamicComp_V4_1
StructDynamicData
StructDynamicFollowing
StructDynamicFollowing_V4_1
StructEffectiveTaskRuntimeType
StructEncoderEffective
StructEncoderMotionState
StructEncoderNumber
StructEncoderSyncCommand
StructEncoderUserDefault
StructFilterForceControl
StructFixedGearingAdjustments
StructFixedGearingOffset
StructFixedGearingSettings
StructFixedGearMotionIn
StructFixedGearMotionOut
StructFixedGearUserDefault
StructFollowingObjectCurrentSyncPosition
StructFollowingObjectDistributedMotion
StructFollowingObjectMasterDynamics
StructFollowingObjectUserDefault
StructForceControllerData2
StructForceControllerData2_V3_1
StructForceControllerData2_V4_1
StructForceControllerDifference
StructForceControlSwitchingData
StructForceControlUserDefault
StructForceLimitingSwitchingData
StructFormulaObjectDINTIn
StructFormulaObjectDINTOut
StructFormulaObjectLREALIn
StructFormulaObjectLREALOut
StructFormulaObjectMotionIn
StructFormulaObjectMotionOut
StructGear

StructGearingAdjustments
StructGearingOffset
StructGearingSettings
StructInternalServoSettings
StructInternalToTrace
StructLimitsOfPathDynamics
StructMeasuringInputEffective
StructMeasuringInputUserDefault
StructMotionVector
StructOutputCamCounterData
StructOutputCamEffectiveData
StructOutputCamUserDefault
StructOutputControllerOutput
StructOutputData
StructOutputLimits
StructOutputMonitoring
StructOutputSettings
StructPathAbortPosition
StructPathAxisMotionVector
StructPathBcsData
StructPathBlending
StructPathCircularCommand
StructPathData
StructPathDynamics
StructPathDynamicsInState
StructPathKinematicsData
StructPathLinearCommand
StructPathMcsData
StructPathMotionActual
StructPathMotionVector
StructPathObjectConnection
StructPathObjectCsFrame
StructPathObjectMotionBuffer
StructPathObjectOcs
StructPathObjectUserDefaultOcs

StructPathOverride
StructPathPolynomialCommand
StructPathSettings
StructPathSpecificVelocityProfile
StructPathUserDefault
StructPathVector
StructPathWSettings
StructPDController
StructPendingAlarmState
structPersistentDataPowerMonitoringtype
StructPID_Controller
StructPID_Controller_V3_1
StructPID_Controller_V4_1
StructPIDController
StructPIDController_V3_1
StructPNSyncCounter
StructProcessModel
StructPVController
StructPVController_V3_1
StructQFAxisDataSet
StructQFAxisDataSet_V3_1
StructQFAxisMaxDerivative
StructQFAxisUserDefault
StructRetAllocateToken
StructRetCommandState
StructRetConfiguration
StructRetDeviceCommandState
StructRetDeviceGetStateOfAllDpStations
StructRetDeviceGetStateOfDpSlave
StructRetDeviceGetStateOfIO
StructRetDeviceNameOfStation
StructRetDiagnosticData
StructRetDpSlaveAddress
StructRetEncoderValue
StructRetGetAxisProgrammedTargetPosition

StructRetGetAxisSpecificState
StructRetGetAxisSpecificState2
StructRetGetAxisStoppingData
StructRetGetCamFollowingDerivative
StructRetGetCamSpecificState
StructRetGetCamSpecificState2
StructRetGetCamTrackSpecificState
StructRetGetCircularPathData
StructRetGetCircularPathData_V4_2
StructRetGetCircularPathGeometricData
StructRetGetConfigurationData
StructRetGetDataByToken
StructRetGetDeviceId
StructRetGetDoIndexNumberFromLogAddress
StructRetGetDpStationAddressFromLogDiagnosticAddress
StructRetGetErrorNumberState
StructRetGetExternalEncoderSpecificState
StructRetGetExternalEncoderSpecificState2
StructRetGetFollowingMinMax
StructRetGetForceControlDataSetParameter
StructRetGetForceControlDataSetParameter_V3_1
StructRetGetForceControlDataSetParameter_V4_1
StructRetGetGearAxisSpecificState
StructRetGetGearAxisSpecificState2
StructRetGetGeoAddressFromLogAddress
StructRetGetInOutByte
StructRetGetLinearPathData
StructRetGetLinearPathData_V4_2
StructRetGetLinearPathGeometricData
StructRetGetLogDiagnosticAddressFromStationAddress
StructRetGetMeasuringInputSpecificState
StructRetGetMeasuringInputSpecificState2
StructRetGetMemoryCardId
StructRetGetNextLogAddress
StructRetGetOutputCamSpecificState

StructRetGetOutputCamSpecificState2
StructRetGetPathGeometricData
StructRetGetPathMotionBuffer
StructRetGetPathObjectBcsFromOcsData
StructRetGetPathObjectOcsFromBcsData
StructRetGetPendingAlarms
StructRetGetPendingAlarmState
StructRetGetPolynomialPathData
StructRetGetPolynomialPathData_V4_2
StructRetGetPolynomialPathGeometricData
StructRetGetPosAxisSpecificState
StructRetGetPosAxisSpecificState2
StructRetGetQFAxisDataSetParameter
StructRetGetQFAxisDataSetParameter_V3_1
StructRetGetSegmentIdentification
StructRetGetStateOfAllDpSlaves
StructRetGetStateOfSingleDpSlave
StructRetGetStateOfTo
StructRetGetStationType
StructRetGetToError
StructRetGetValue
StructRetIPConfig
StructRetMotionBuffer
StructRetMotionCommandState
StructRetPathAxesData
StructRetPathAxesPosition
StructRetPathCartesianData
StructRetPathCartesianPosition
StructRetPathMotionCommandState
StructRetReadDriveFaults
StructRetReadDriveMultiParameter
StructRetReadDriveMultiParameterDescription
StructRetReadDriveParameter
StructRetReadDriveParameterDescription
StructRetReadGetAxisDataSet

StructRetReadGetAxisDataSet_V3_1
StructRetReadGetAxisDataSet_V4_1
StructRetReadRecord
StructRetTcpOpenClient
StructRetTcpOpenServer
StructRetTcpReceive
StructRetUdpReceive
StructRetUnitDataSetCommand
StructRetWriteDriveMultiParameter
StructRetWriteDriveParameter
StructRetXCommandState
StructRetXreceive
StructScaleOffset
StructSensorMonitorings
StructStateOfDpStations
StructStateOfIO
StructSyncDynamics
StructSyncMonitoring
StructSyncPositions
StructSyncProfileDefinitions
StructSystemLoad
StructTaskId
StructTaskOverflowType
StructTaskRuntimeType
StructTaskRuntimeValues
StructTCOFctGenOverride
StructTControllerActualDPIDData
StructTControllerActualIdentificationModifiedTangentMethodData
StructTControllerActualIdentificationStandardTangentMethodData
StructTControllerActualInputData
StructTControllerActualInputLimitCheckData
StructTControllerActualInputSingleLimitCheckData
StructTControllerControlRangeParameter
StructTControllerCycleParameter
StructTControllerDPIDParameter

StructTControllerIdentificationModifiedTangentMethodParameter
StructTControllerIdentificationModifiedTangentMethodProcessParameter
StructTControllerIdentificationStandardTangentMethodParameter
StructTControllerIdentificationStandardTangentMethodProcessParameter
StructTControllerIdentificationStaticCondition
StructTControllerInputDisplayValueParameter
StructTControllerInputFilterParameter
StructTControllerInputGradientCheckParameter
StructTControllerInputLimitCheckParameter
StructTControllerInputSingleLimitCheckParameter
StructTControllerOutputValue
StructTControllerPlausibilityParameter
StructTControllerProcessModeParameter
StructTControllerPWMParameter
StructTorqueLimit
StructTraceControl
StructTraceState
StructUserData
StructValueAndDerivedMasterValue
StructValueAndDerivedSlaveValue
StructVelocityGearingSettings
StructXsendDestAddr
SUB
SUB_DATE_DATE
SUB_DT_DT
SUB_DT_TIME
SUB_TIME
SUB_TOD_TIME
SUB_TOD_TOD
SUPERIMPOSED_MOTION
SUPERIMPOSED_MOTION_ACTIVE
SUPERIMPOSED_MOTION_MERGE
SUPERIMPOSED_POS_MOTION_ACTIVE
superimposedMotion
SWITCH_BY_VALUE

SWITCHING_CONDITION_DONE
swLimit
swLimitState
SYMBOL_INFORMATION_NOT_AVAILABLE
SYMBOL_INFORMATION_NOT_FOUND
syncCommand
SYNCHRONIZE_SYMMETRIC
SYNCHRONIZE_WHEN_POSITION_REACHED
SYNCHRONIZED
SYNCHRONIZING
SYNCHRONIZING_NOT_POSSIBLE
synchronizingState
syncMonitoring
syncState
SYSTEM
SYSTEM_DEFINED
systemClock
systemLoad
SYSVAR

T

TAN
TARGET_POSITION_MODE
TASK_EXECUTION
TASK_STATE_INVALID
TASK_STATE_LOCKED
TASK_STATE_RUNNING
TASK_STATE_STOP_PENDING
TASK_STATE_STOPPED
TASK_STATE_SUSPENDED
TASK_STATE_WAIT_NEXT_CYCLE
TASK_STATE_WAIT_NEXT_INTERRUPT
TASK_STATE_WAITING
taskRuntime
taskRuntimeMonitoring

TCOFctGenOverride
TemperatureControllerType
TEMPORARY_STORAGE
THEN
TIME
TIME_AND_INPUT
TIME_CONDITION
TIME_LOCKED_PROFILE
TIME_OF_DAY
TIME_OR_INPUT
TIME_OUT
TIME_TO_INT
TIME_TO_REAL
TIME_TO_UDINT
TO
TO_CONNECTION
TO_EXECUTION
TOD
TOF
TON
TORQUE
torqueLimitingCommand
torqueLimitNegative
torqueLimitNegativeIn
torqueLimitPositive
torqueLimitPositiveIn
TOTAL_MOVE
tOutput
TP
TRACE_ABORTED
TRACE_FINISHED
TRACE_INACTIVE
TRACE_MISMATCH
TRACE_NO_TIME
TRACE_PARAM_VALID

TRACE_RUNNING
TRACE_WAIT_FOR_TRIGGER
traceControl
traceState
TRANSFER
TRANSIENT_BEHAVIOR_DIRECT
TRANSIENT_BEHAVIOR_WITH_DYNAMICS
TRANSIENT_BEHAVIOR_WITH_NEXT_SYNC
TRANSITION
TRAPEZOIDAL
TRIGGER_OCCURED
TRUE
TRUNC
TYPE
typeOfAxis

U

UDINT
UDINT_TO_BYTE
UDINT_TO_DINT
UDINT_TO_DWORD
UDINT_TO_INT
UDINT_TO_LREAL
UDINT_TO_REAL
UDINT_TO_SINT
UDINT_TO_STRING
UDINT_TO_TIME
UDINT_TO_UINT
UDINT_TO_USINT
UDINT_TO_WORD
UDINT_VALUE_TO_BOOL
UINT
UINT_TO_BYTE
UINT_TO_DINT
UINT_TO_DWORD

UINT_TO_INT
UINT_TO_LREAL
UINT_TO_REAL
UINT_TO_SINT
UINT_TO_UDINT
UINT_TO_USINT
UINT_TO_WORD
UINT_VALUE_TO_BOOL
ULINT
UNDER
UNI_DIRECTION
UNIT
UNIT_NOT_FOUND
UNTIL
UPPER_PRIMARY_FAILED
UPPER_SECONDARY_FAILED
upperPlausibilityParameter
upperPlausibilityParameterSecondary
USELIB
USEPACKAGE
USER
USER_DEFAULT
USER_DEFINED
USER_EVENTS_1
USER_EVENTS_2
USER_STORAGE
userData
userDefault
userDefaultClamping
userDefaultDynamics
userDefaultForceControl
userDefaultForceLimiting
userDefaultHoming
userDefaultOcs
userDefaultPositioning

userDefaultQFAxis
userDefaultTorqueLimiting
USES
USINT
USINT_TO_BYTE
USINT_TO_DINT
USINT_TO_DWORD
USINT_TO_INT
USINT_TO_LREAL
USINT_TO_REAL
USINT_TO_SINT
USINT_TO_UDINT
USINT_TO_UINT
USINT_TO_WORD
USINT_VALUE_TO_BOOL

V

VALID
VALUE
VALUE_LEFT_MARGIN
VALUE_LEFT_MARGIN_WITHOUT_SIGN
VALUE_RIGHT_MARGIN
VALUE_RIGHT_MARGIN_WITHOUT_SIGN
VAR
VAR_ACCESS
VAR_ALIAS
VAR_EXTERNAL
VAR_GLOBAL
VAR_IN_OUT
VAR_INPUT
VAR_OBJECT
VAR_OUTPUT
VAR_TEMP
VELOCITY
VELOCITY_BASED_LIMIT

VELOCITY_CONTROLLED
VELOCITY_GEARING
velocityBasedMotionInCommand
velocityLimitingCommand
velocityMotionInPositionProfileCommand
velocityPositionProfileCommand
velocityTimeProfileCommand
VERSION_MISSMATCH
VIRTUAL_AXIS
VOID

W

WAIT_FOR_CONDITION
WAIT_FOR_DP_CLOCK
WAIT_FOR_VALID
WAITFORCONDITION
WAITING
WAITING_FOR_CHANGE_OF_MASTER_DIRECTION
WAITING_FOR_MERGE
WAITING_FOR_RESTART
WAITING_FOR_SYNC_POSITION
WAITING_FOR_SYNC_START
WAITING_FOR_TRACKING_START
WAITING_FOR_TRIGGER
WAITING_TO_START
WARNING
WHEN_ACCELERATION_DONE
WHEN_AXIS_HOMED
WHEN_AXIS_SYNCHRONIZED
WHEN_BUFFER_READY
WHEN_CAM_TRACK_DONE
WHEN_COMMAND_DONE
WHEN_DATA_ACTIVE
WHEN_ENCODER_SYNCHRONIZED
WHEN_ENDSTOP_REACHED

WHEN_FORCE_CONTROL_ENABLED
WHEN_FORCE_LIMIT_REACHED
WHEN_FORCE_LIMITING_ACTIVATED
WHEN_FUNCTION_DISABLED
WHEN_GEARING_START
WHEN_INTERPOLATION_DONE
WHEN_LIMIT_REACHED
WHEN_LIMITING_COMMAND_ACTIVATED
WHEN_MOTION_DONE
WHEN_PROFILE_DONE
WHEN_TORQUELIMIT_GONE
WHEN_TORQUELIMIT_REACHED
WHEN_TRIGGER_OCCURED
WHILE
WITH
WITH_CAM_TRACK_ACTIVATION
WITH_COMMAND_VALUE_ZERO
WITH_DYNAMICS
WITH_INTERPOLATION
WITH_MAXIMAL_DECELERATION
WITH_NEXT_SYNCHRONIZING
WITH_RADIUS_AND_ENDPOSITION
WITH_SPECIFIC_AREA
WITH_TIME_LIMIT
WITHOUT_CHANGE
WITHOUT_INTERPOLATION
WITHOUT_INTERPOLATION_AND_HOLD_ACTIVE_CAM
WITHOUT_LIMITING
WITHOUT_SPECIFIC_AREA
WITHOUT_TIME_LIMIT
WORD
WORD_TO_BCD
WORD_TO_BOOL
WORD_TO_BYTE
WORD_TO_DINT

WORD_TO_DWORD
WORD_TO_INT
WORD_TO_SINT
WORD_TO_UDINT
WORD_TO_UINT
WORD_TO_USINT
WORD_VALUE_TO_LREAL
WORD_VALUE_TO_REAL
WRITE_JOBS
WRITEABLE

X

X_Y
X_Y_Z
XOR
XOR
XORN

Y

Y_Z
YES

Z

Z_X
ZERO_VALUE
ZM_PASSIVE

A.3 Assignment types for symbolic assignment

A.3.1 Interface types of the technology objects

Description

SIMOTION technology objects and SINAMICS objects provide interfaces and assignment types for symbolic assignment; these are described in detail below.

Only interfaces of the same type can be assigned.

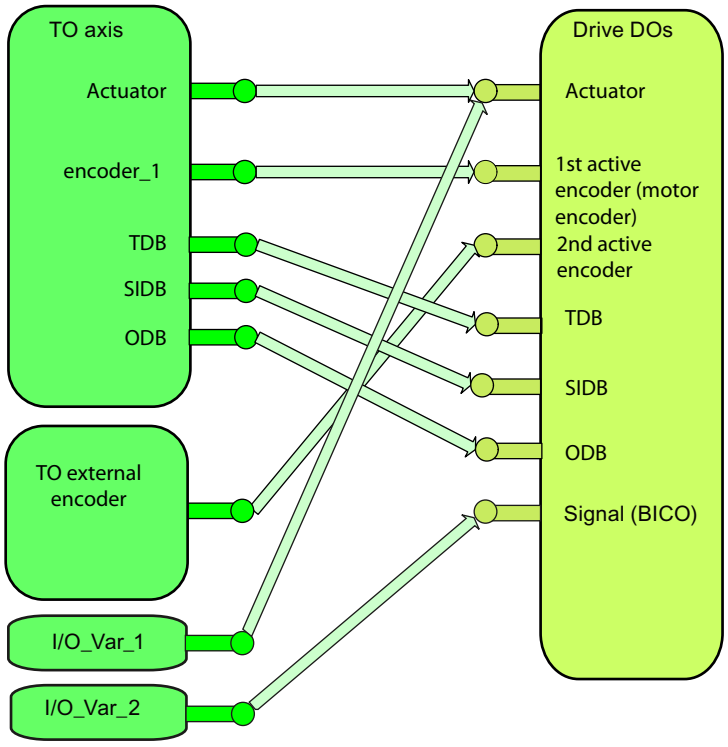


Figure A-1 Example TO - DO assignment

Assignment types for the Axis TO

Assignment partner designation	Assignment type	Description
Actuator	• ActorProfidrive • ActorOnboard • WordOutput	Axis setpoint interface • Via PROFIdrive message frame • Onboard • Analog output for hydraulics
Technology data	TDB	Technology data on the axis
Safety data	SIDB	Safety data on the axis
encoder_<n>	BitInput	Encoder interface of the nth encoder

Assignment partner designation	Assignment type	Description
StatusExternalZeroMark	BitInput	External zero mark status of a digital drive. For incremental encoders
Homing output cam	BitInput	Homing output cam
HomingOutputCamPassive-Homing	BitInput	Homing output cam for passive homing
Ready status detection	BitInput	Ready status detection for analog sensor
Error status detection	BitInput	Error status detection for analog sensor
Update counter	BitInput	Update counter, analog sensor
Positive reversing cam	BitInput	Positive reversing cam
Negative reversing cam	BitInput	Negative reversing cam
HW limit switch positive	BitInput	Hardware limit switch positive
HW limit switch negative	BitInput	Hardware limit switch negative
Q-valve	WordOutput	Q output for hydraulics, Q-valve
Q-valve enable	BitOutput	Enable bit for Q-valve
P-valve	WordOutput	F output for hydraulics, P-valve
P-valve enable	BitOutput	Enable bit for P-valve
Pressure sensor	Wordinput	Pressure sensor
Digital input	BitInput	Digital input for fast switching to pressure control

Assignment types for the External Encoder TO

Assignment partner designation	Assignment type	Description
encoder_1	• sensorPROFIdrive • sensorOnboard • wordInput • dwordInput	Encoder interface of the encoder
StatusExternalZeroMark	bitInput	External zero mark status of a digital drive. For incremental encoders
Homing output cam	bitInput	Homing output cam
HomingOutputCamPassive-Homing	bitInput	Homing output cam for passive homing
Ready status detection	bitInput	Ready status detection for analog sensor
Error status detection	bitInput	Ready status detection for analog sensor
Update counter	bitInput	Update counter, analog sensor
Measuring input		Measuring input input for local measuring

Assignment types for the Output Cam TO

Assignment partner designation	Assignment type	Description
Output	• bitOutput • bitOutputWithTimeStamp	Output cam output

Assignment types for the Measuring Input TO

Assignment partner designation	Assignment type	Description
Input	bitInputWithTimeStamp	Measuring input input

Assignment types for the TControl TO

The following interface types are available on the TO sensor:

Assignment partner designation	Assignment type	Description
Temperature input	wordInput	Analog input, temperature
PWM signal	bitInput	Output, PWM signal
PWM signal_2	bitInput	Output, PWM signal_2

Assignment types for the Sensor TO

The following interface type is available on the TO sensor:

Assignment partner designation	Assignment type	Description
Analog sensor	wordInput	Analog input

A.3.2 SINAMICS interface types

Description

The various SINAMICS DOs offer the following I/O interfaces to SIMOTION interfaces.

DO servo, vector, and TM41 (drive DOs)

Name	Interface type	Description
Actor	actorProfidrive	Axis setpoint interface (via PROFIdrive message frame)
Encoder_1	sensorPROFIdrive	Encoder 1 on the drive

Name	Interface type	Description
Encoder_2	sensorPROFIdrive	Encoder 2 on the drive
Bico_iw.<BICO> Bico_id.<BICO>	wordInputDriveParameter	Additional parameter transferred via BICO interconnection (input side)
Bico_qw.<BICO> Bico_qd.<BICO>	dwordInputDriveParameter	Additional parameter transferred via BICO interconnection (output side)
For information on interfaces, see: TM41 inputs/outputs in the SCOUT online help.		

Encoder DO

Name	Interface type	Description
encoder	sensorPROFIdrive	PROFIdrive encoder interface
Bico_iw.<BICO> Bico_id.<BICO>	wordInputDriveParameter	Additional parameter transferred via BICO interconnection (input side)
Bico_qw.<BICO> Bico_qd.<BICO>	dwordInputDriveParameter	Additional parameter transferred via BICO interconnection (output side)
For information on interfaces, see: Encoder DO in the SCOUT online help.		

DO TM15/TM17 High Feature

Name	Interface type	Description
DI_<n> MI_<n> DO_<n> CAM_<n>	bitInput bitInputWithTimeStamp bitOutput bitOutputWithTimeStamp	Interfaces for DIs, DOs, measuring inputs, output cams for terminals X520, X521, X522; interface type depends on terminal configuration TM15: 24 channels, TM17: up to 16 channels
DI_0_15 DI_16_23 DO_0_15 DO_16_23	_wordInputInterfaceType wordOutputInterfaceType	Group interfaces for replacement values (Wordinput, Wordoutput)
For information on interfaces, see: TM17 inputs/outputs TM15 inputs/outputs in the SCOUT online help.		

DO control unit

Name	Interface type	Description
DI_<n>	bitInput	Digital inputs of the control unit with SIMOTION D, CX32/CX32-2, and SINAMICS S110/S120 control units
DI_<n> MI_<n> DO_<n> CAM_<n>	bitInput bitInputWithTimeStamp bitOutput bitOutputWithTimeStamp	Bidirectional digital inputs/outputs DI interfaces (bidirectional digital input) Measuring input input DO interfaces (bidirectional digital output) Output of output cam for the terminals

Name	Interface type	Description
DI_0_15 DO_0_15	_wordInputInterfaceType _wordOutputInterfaceType	Group interfaces for replacement values (Wordinput, Wordoutput)
Bico_iw.<BICO> Bico_id.<BICO>	wordInputDriveParameter	Additional parameter transferred via BICO interconnection (input side).
Bico_qw.<BICO> Bico_qd.<BICO>	wordOutputDriveParameter	Additional parameter transferred via BICO interconnection (output side)
For information on interfaces, see: Control Unit CU320 inputs/outputs in the SCOUT online help.		

Additional SINAMICS DO (such as TM31, TB30, and TM15 DI/DO)

Name	Interface type	Description
Interfaces depending on the standard message frame		
Bico_iw.<BICO>	WordInputDriveParameter	Additional parameter transferred via BICO interconnection (input side). The number of interconnectable parameters depends on the message frame used.
Bico_qw.<BICO>	WordOutputDriveParameter	Additional parameter transferred via BICO interconnection (output side). The number of interconnectable parameters depends on the message frame used.

A.3.3 Interfaces in standard message frames

Description

Standard message frames are standardized and transfer a set number of interfaces with certain properties.

They contain the following interfaces:

Standard message frame 1

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_A	WORD	Actor interface, speed setpoint A, 16 bits
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_A	WORD	Actor interface, actual speed value A, 16 bits

Standard message frame 2

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2

Standard message frame 3

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2

Standard message frame 4

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
encoder_2.Gn_STW	WORD	Encoder 2, control word
encoder_2.Gn_ZSW	WORD	Encoder 2, status word
encoder_2.Gn_XIST1	DWORD	Encoder 2, act. position val. 1
encoder_2.Gn_XIST2	DWORD	Encoder 2, act. position val. 2

Standard message frame 5

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
actor.XERR	DWORD	Actor interface, position deviation
actor.KPC	DWORD	Actor interface, position controller gain factor

Standard message frame 6

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
actor.MOMRED	WORD	Actor interface, torque reduction
actor.MELDW	WORD	Actor interface, message word
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
encoder_2.Gn_STW	WORD	Encoder 2, control word
encoder_2.Gn_ZSW	WORD	Encoder 2, status word
encoder_2.Gn_XIST1	DWORD	Encoder 2, act. position val. 1
encoder_2.Gn_XIST2	DWORD	Encoder 2, act. position val. 2

SIEMENS message frame 101

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1

Name	Data type	Description
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
actor.MOMRED	WORD	Actor interface, torque reduction
actor.MELDW	WORD	Actor interface, message word

SIEMENS message frame 102

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
actor.MOMRED	WORD	Actor interface, torque reduction
actor.MELDW	WORD	Actor interface, message word
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2

SIEMENS message frame 103

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
actor.MOMRED	WORD	Actor interface, torque reduction
actor.MELDW	WORD	Actor interface, message word
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
encoder_2.Gn_STW	WORD	Encoder 2, control word
encoder_2.Gn_ZSW	WORD	Encoder 2, status word
encoder_2.Gn_XIST1	DWORD	Encoder 2, act. position val. 1
encoder_2.Gn_XIST2	DWORD	Encoder 2, act. position val. 2

SIEMENS message frame 105

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
actor.MOMRED	WORD	Actor interface, torque reduction
actor.MELDW	WORD	Actor interface, message word
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
actor.XERR	DWORD	Actor interface, position deviation
actor.KPC	DWORD	Actor interface, position controller gain factor

SIEMENS message frame 106

Name	Data type	Description
actor.STW1	WORD	Actor interface, control word 1
actor.NSOLL_B	DWORD	Actor interface, speed setpoint B, 32 bits
actor.STW2	WORD	Actor interface, control word 2
actor.ZSW1	WORD	Actor interface, status word 1
actor.NIST_B	DWORD	Actor interface, actual speed value B, 32 bits
actor.ZSW2	WORD	Actor interface, status word 2
actor.MOMRED	WORD	Actor interface, torque reduction
actor.MELDW	WORD	Actor interface, message word
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
encoder_2.Gn_STW	WORD	Encoder 2, control word
encoder_2.Gn_ZSW	WORD	Encoder 2, status word
encoder_2.Gn_XIST1	DWORD	Encoder 2, act. position val. 1
encoder_2.Gn_XIST2	DWORD	Encoder 2, act. position val. 2
actor.XERR	DWORD	Actor interface, position deviation
actor.KPC	DWORD	Actor interface, position controller gain factor

Standard message frame 81

Name	Data type	Description
encoder_1.STW2	WORD	Control word 2
encoder_1.ZSW2	WORD	Status word 2
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2

Standard message frame 82

Name	Data type	Description
encoder_1.STW2	WORD	Control word 2
encoder_1.ZSW2	WORD	Status word 2
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
encoder_1.NIST_A	WORD	Encoder 1, actual speed value A, 16 bits

Standard message frame 83

Name	Data type	Description
encoder_1.STW2	WORD	Control word 2
encoder_1.ZSW2	WORD	Status word 2
encoder_1.Gn_STW	WORD	Encoder 1, control word
encoder_1.Gn_ZSW	WORD	Encoder 1, status word
encoder_1.Gn_XIST1	DWORD	Encoder 1, act. position val. 1
encoder_1.Gn_XIST2	DWORD	Encoder 1, act. position val. 2
encoder_1.NIST_B	DWORD	Encoder 1, actual speed value B, 32 bits

Structure of message frame extension TDB

Name	Data type	Description
M_ADD	WORD	Additional torque
M_LIMIT_PLUS	WORD	Positive torque limit
M_LIMIT_MINUS	WORD	Negative torque limit
M_ACTUAL	WORD	Current torque

Structure of message frame extension SIDB

Name	Data type	Description
S_ZSW1B	WORD	Safety status word
S_SPEED_LIMIT	DWORD	Velocity limit signal

A.3.4 Designations for interface types

Description

The following assignment types have been defined for SIMOTION:

Axis and encoder interface types

Table A-6 Axis interface types

Name under ST nomenclature	Name in the address list
_actorProfidriveInterfaceType	ActorProfidrive
_actorOnboardInterfaceType	ActorOnboard
_wordOutputInterfaceType	WordOutput

Table A-7 Encoder interface types

Name under ST nomenclature	Name in the address list
_sensorProfidriveInterfaceType	SensorProfidrive
_sensorOnboardInterfaceType	SensorOnboard
_wordInputInterfaceType	WordInput
_dwordInputInterfaceType	DWordInput

Table A-8 Data blocks (extended communication)

Name under ST nomenclature	Name in the address list
_tdbInterfaceType	TDB
_sidbInterfaceType	SIDB

Output cam, measuring input

Name in ST naming scheme of the output cam assignment types	Name in the address list
_bitOutputInterfaceType	BitOutput
_bitOutputWithTimeStampInterfaceType	BitOutputWithTimeStamp

Name in ST naming scheme of the measuring input assignment types	Name in the address list
_bitInputWithTimeStampInterfaceType	BitInputWithTimeStamp

Standard I/O interfaces

Name under ST nomenclature	Name in the address list
_bitInputInterfaceType	BitInput
_bitOutputInterfaceType	BitOutput
_byteInputInterfaceType	ByteInput
_byteOutputInterfaceType	ByteOutput
_wordInputInterfaceType	WordInput
_wordOutputInterfaceType	WordOutput
_dwordInputInterfaceType	DWordInput
_dwordOutputInterfaceType	DWordOutput

SINAMICS parameter interface types

Name under ST nomenclature	Name in the address list
_wordInputDriveParameterInterfaceType	WordInputDriveParameter
_dwordInputDriveParameterInterfaceType	DWordInputDriveParameter
_wordOutputDriveParameterInterfaceType	WordOutputDriveParameter
_dwordOutputDriveParameterInterfaceType	DWordOutputDriveParameter

A.3.5 Syntax of the assignments

You can also enter the symbolic assignments for the I/O variable using text fields in the address list and TO configuration dialogs (in addition to the buttons for calling the assignment dialog). The syntax required for this purpose is specified in the table below.

Table A-9

Interconnection target	Syntax
Onboard I/Os of SIMOTION D4x5/ D4x5-2, terminal X122/X132 (SINAMICS Integrated);	<device>.<DO_name>.<interface_name> [<terminal_name>] **Example of D4x5-2:** SINAMICS_Integrated.Control_Unit.DO_8 [DI/DO 8, X122.9] -> Digital output "DO_8" on "SINAMICS_Integrated" device, "Control_Unit" drive object, terminal "X122.9", pin designation "DI/DO 8"
Onboard I/Os of SIMOTION D4x5-2, terminal X142	<device>.<<interface_name> **Example:** D455-2, X142: D455.CAM_6 [DI/DO 6, X142.12] -> Output with output cam quality "CAM_6" on "D455" device, terminal "X142.12", pin designation "DI/DO 6"
Onboard I/Os of SINAMICS S120 CU3xx and CX32/32-2 controller extension	<device>.<DO_name>.<interface_name> [<terminal_name>] **Example:** SIMOTION_CX32.Control_Unit.DI_1 [DI 1, X122.2] S120_CU320.Control_Unit.DI_1 [DI 1, X122.2] -> Digital input "DI_1" on "SIMOTION_CX32" or "S120_CU320" device, "Control_Unit" drive object, terminal "X122.2", pin designation "DI 1"
TM15, TM17, TM41	<device>.<DO_name>.<interface_name> [<terminal_name>] **Examples:** TM15: SINAMICS_Integrated.TM15.MI_8 [DI/DO 8, X521.2] -> Input with measuring input quality "MI_8" on "SINAMICS_Integrated" device, "TM15" drive object, terminal "X521.2", pin designation "DI/DO 8" TM41: SINAMICS_Integrated.TM41.AI_0 [AI 0, X523] -> Analog input "AI_0" on "SINAMICS_Integrated" device, "TM41" drive object, terminal "X523", pin designation "AI 0"

Interconnection target	Syntax
TB30, TM31, TM15 DI/DO	<device>.<DO_name>.<interface_name> [<terminal_name>] **Examples:** TB30: SINAMICS_Integrated.TB30.AI_0 [AI 0, X482.1, X482.2] -> Analog input "AI_0" on "SINAMICS_Integrated" device, "TB30" drive object, terminals "X482.1" and "X482.2", pin designation "AI 0" TM31: SINAMICS_Integrated.TM31.DI_2 [DI 2, X520.3] -> Digital input "DI_2" on "SINAMICS_Integrated" device, "TM31" drive object, terminal "X520.3", pin designation "DI 2" TM15 DI/DO: SINAMICS_Integrated.TM15DIDO.DO_23 [DI/DO 23, X522.9] -> Digital output "DO_23" on "SINAMICS_Integrated" device, "TM15DIDO" drive object, terminal "X522.9", pin designation "DI/DO 23"
SINAMICS free message frame extensions	<device>.<DO_name>.BICO_xx.<parameter_number> with (xx = IW, QW, ID, QD) for interconnection to a parameter in the same drive object (DO), to which the message frame is addressed <device>.<DO_name>. BICO_xx.<DO_name2>#<parameter_number> for interconnection to a parameter in drive object (DO) <DO_name2>, whereby the message frame is configured for DO <DO_name>
Message frame components	<device>.<DO_name>.<interface_name>.<subinterface_name> **Examples:** SINAMICS_Integrated.Axis_Red.actor.STW1 SINAMICS_Integrated.Axis_Blue.actor.ZSW1 SINAMICS_Integrated.Axis_Red.encoder_1.Gx_ZSW
Message frame of the infeed unit	If the automatic message frame determination is activated, the message frame for the infeed is specified automatically (PROFIdrive message frame 370). You can now create two I/O variables in WORD format via symbolic assignment in the address list for the status and control words (ZSW1/STW1) of the infeed. You require these I/O variables for the **FB _LineModule_control** for the control of the line module. For further information refer to the *Standard Function for SINAMICS S120 Line Modules* Function Manual
Onboard I/Os of SIMOTION C	<device>.<interface_name> **Examples:** C240_1.AnalogActor_3 [ANALOG/STEPPER 3, X2] C240_1.Encoder_2 [ENCODER 2, X4] C240_1.DI_0 [I 0, X1.28] C240_1.DO_7 [Q7, X.19] C240_1.M1 [M1, X1.26] C240_1.B2 [B2, X1.23]

Index

_
_acknowledgeAlarmSqId, 369
_acknowledgeAllAlarmSqId, 371
_AdditionObjectType, 130
_alarm, 356
_alarmScId
  Application, 471, 472
  Description, 362
_alarmSId
  Application, 471, 472
  Description, 357
_alarmSqId
  Application, 471, 472
  Description, 359
_AND, 384
_ArrayCopy, 465
_BYTE_FROM_8BOOL, 404
_BYTE_TO_8BOOL, 539
_CamTrackType, 130
_checkEqualTask
  Description, 348
_checkExistingDataSet
  Application, 481
  Description, 443
_ControllerObjectType, 130
_deleteAllUnitDataSets
  Application, 481
  Description, 445
_deleteUnitDataSet
  Application, 481
  Description, 440
_device, 429, 432, 441, 444, 446, 482
_directoryPathDelete, 497
_disableScheduler
  Brief description, 339
_DWORD_FROM_2WORD, 406
_DWORD_FROM_4BYTE, 407
_DWORD_TO_2WORD, 541
_DWORD_TO_4BYTE, 541
_enableScheduler
  Brief description, 339
_exportUnitDataSet
  Application, 481
  Description, 434
_fileClose, 489
_fileCopy, 497
_fileDelete, 497
_filehandle, 488
_fileOpen, 488
_fileRead, 488
_fileReadLn, 488
_fileRename, 497
_fileSetPosition, 488
_fileWrite, 488
_fileWriteLn, 489
_finite
  Description, 408
_firstIndexOf, 457
_FixedGearType, 130
_FormulaObjectType, 130
_getAlarmId
  Application, 356
  Description, 364
_getAverageTaskIdRunTime, 353
_getBit, 378
_getCommandId, 448
  Error source, 592
_getCurrentTaskIdRunTime, 352
_getDeviceId, 451
_getInOutByte
  Description, 426
_getInternalTimeStamp, 354
_getLogicalAddressOfIoVariable, 464
_getMaximalTaskIdRunTime, 350
_getMemoryCardId, 452
_getMinimalTaskIdRunTime, 351
_getPendingAlarms, 365
_getsafeValue, 420
_getStateOfFile, 497
_getStateOfTaskId
  Brief description, 339
  Description, 341
_getStateOfUnitDataSetCommand
  Description, 442
  Example, 486
_GetStateOfXCommand, 507
_getSyncCommandId, 448
  Application, 514
_getTaskId
  Application, 339
  Description, 347
_getTimeDifferenceOfInternalTimeStamps, 355
_importUnitDataSet
  Application, 481
  Description, 437

_isNaN
Description, 409
_lastIndexOf, 458
_lengthIndexOf, 459
_loadUnitDataSet
Application, 481
Description, 431
Example, 485
_NOT, 384
_OR, 384
_PathAxis, 130
_Pathobjecttype, 130
_project, 140
See also name space, 140
_REFl, 469
_releaseSemaphore
Application, 479
Description, 419
_resetAlarmId, 366
_resetAllAlarmId, 368
_resetTaskId
Brief description, 338
Description, 342
_resetUnitData
Application, 481
_restartTaskId
Brief description, 338
Description, 343
_resumeTaskId
Brief description, 339
Description, 343
_retriggerTaskIdControlTime
Brief description, 339
Description, 344
_S7_COUNTER, 543
_S7_TIMER, 544
_saveUnitDataSet
Application, 481
Description, 428
Example, 486
_SC_ALARM_CONFIGURATION, 178
_SC_ARRAY_BOUND_ERROR_READ, 166
_SC_ARRAY_BOUND_ERROR_WRITE, 166
_SC_BACKGROUND_TIMER_OVERFLOW, 177
_SC_CLASS_INSTANCE_NOT_EXISTENT, 166
_SC_CYCLE_TIMER_OVERFLOW, 177
_SC_DEVICE_COMMAND, 178
_SC_DIAGNOSTIC_INTERRUPT, 167
_SC_DIVISION_BY_ZERO, 166
_SC_DP_CLOCK_DETECTED, 168
_SC_DP_SLAVE_NOT_SYNCHRONIZED, 168
_SC_DP_SLAVE_SYNCHRONIZED, 168
_SC_DP_SYNCHRONIZATION_LOST, 168
_SC_DRIVE_OBJECT_, 169
_SC_DRIVE_OBJECT_ALARM, 169
_SC_EXCEPTION, 178
_SC_EXTERNAL_COMMAND, 178
_SC_IMAGE_UPDATE_FAILED, 168
_SC_IMAGE_UPDATE_OK, 168
_SC_INVALID_ADDRESS, 170
_SC_INVALID_FLOATING_POINT_OPERATION, 166
_SC_IO_MODULE_NOT_SYNCHRONIZED, 169
_SC_IO_MODULE_SYNCHRONIZED, 169
_SC_MODE_SELECTOR, 178
_SC_PC_INTERNAL_FAILURE, 168
_SC_PROCESS_INTERRUPT, 167
_SC_PULL_PLUG_INTERRUPT, 169
_SC_STATION_DISCONNECTED, 167
_SC_STATION_RECONNECTED, 168
_SC_TASK_STACKSIZE_FAULT, 166
_SC_TO_INSTANCE_NOT_EXISTENT, 166
_SC_VARIABLE_ACCESS_ERROR_READ, 166
_SC_VARIABLE_ACCESS_ERROR_WRITE, 166
_SensorType, 130
_setBit, 379
_setDeviceErrorLED, 453
_setDriveObjectSTW, 454
_setSafeValue
Description, 422
_sizeOf, 456
_startSyncCommand
Application, 514
_startTaskId
Brief description, 338
Description, 345
_suspendTaskId
Brief description, 338
Description, 346
_task, 339
_testAndSetSemaphore
Application, 479
Description, 418
_to, 140
see also Name space, 140
_toggleBit, 382
_trcVal, 467
_waitTime, 592
Application, 337
Description, 450
_WORD_FROM_2BYTE, 405
_WORD_TO_2BYTE, 540
_writeAndSendMessage
device function, 471
_XOR, 384

_Xreceive
Application, 504
Parameter description, 507
_Xsend
Application, 504
Parameter description, 505
Structure of destination address, 505

A

ABS, 374
ACOS, 375
Actual torque, 66
Adaptation, 87
Addition object, 46
Additive torque, 67
Alarm group association, 180
AlarmId, 356
Alarms
Configuration, 153
Possible reactions, 153
Querying TaskStartInfo, 178
AlarmS
Buffer management, 473
ALARMS_ERROR, 358, 359, 361, 362, 363
ALARMS_QSTATE, 363
ALARMS_STATE, 363
ANYOBJECT, 130, 408
AnyObject_to_Object
Description, 408
AnyType_to_BigByteArray
Description, 396
AnyType_to_LittleByteArray
Description, 396
Application cycle clock
Second, 264
ASIN, 375
Assignment destination
Syntax, 708
Assignment dialog
I/O variables, 100
Using, 105
Assignment of axis and drive, 87
Assignment types
Description, 706
SINAMICS DO, 698
Standard message frame, 700
Technology objects, 696
Assignments
Syntax, 708
Asynchronous command execution, 37
For data transmission, 504
ATAN, 375
Axis, 45
Drive axis, 130
Following axis, 130
Position axis, 130
Axis array, 601

B

BackgroundTask, 201, 218
TaskStartInfo, 162
Basic cycle clock, 198
BEGIN_SYNC
Application, 514
BigByteArray_to_AnyType
Description, 398
Bistable function block, 525
Bit string standard functions, 376
Blocking calls, 592
BOOL_TO_BYTE, 388
BOOL_TO_DWORD, 388
BOOL_TO_WORD, 388
BOOL_VALUE_TO_DINT, 388
BOOL_VALUE_TO_INT, 388
BOOL_VALUE_TO_REAL, 388
BOOL_VALUE_TO_SINT, 388
BOOL_VALUE_TO_UDINT, 388
BOOL_VALUE_TO_UINT, 388
BOOL_VALUE_TO_USINT, 388
Buffer management
AlarmS messages, 473
Bus cycle clock, 198
BYTE_TO_BOOL, 388
BYTE_TO_DINT, 388
BYTE_TO_DWORD, 388
BYTE_TO_INT, 388
BYTE_TO_SINT, 389
BYTE_TO_UDINT, 389
BYTE_TO_UINT, 389
BYTE_TO_USINT, 389
BYTE_TO_WORD, 389
BYTE_VALUE_TO_LREAL, 389
BYTE_VALUE_TO_REAL, 389

C

CACF, 266
Cam, 45, 130
Cam track, 45
CamType, 130
CLib Studio, 25

Clock synchronization
Terminal-terminal, 292
Command processing
Diagnostics, 138
Command reference - commandId;, 137
CommandId, 37
Error source, 592
Commands
CommandId, 37
Return value, 37, 122, 192
Step enabling condition; command enabling condition, 133
synchronous/asynchronous, 37
Communication commands, 504
Comparing
Expert list, 53, 56
CONCAT_DATE_TOD, 392
Configuration, 36
Execution system, 244
Configuration data, 36
Consistency commands
Application, 478
Description, 417
Constant bus cycle time, 249
Controller object, 46
ControlPanelTask, 202
COS, 375
CPU
STOP, 153
CTD, 532
CTD_DINT, 532
CTD_UDINT, 533
CTU, 530
CTU_DINT, 531
CTU_UDINT, 531
CTUD, 534
CTUD_DINT, 534
CTUD_UDINT, 535
Cycle clock source
Selection, 249
Cycle time
Monitoring, 263
Cyclic program execution
Determining, 320
Influencing, 338
Status of the task, 327
Timeout, 592
Cyclic tasks, 201

D

Data
Sending, 504
Data processing
Isochronous, 285
Data types
Conversions, 387
Enumeration, 128
Enumerators, 128
Error sources, 591
Technology objects, 130
DATE_AND_TIME_TO_DATE, 392
DATE_AND_TIME_TO_TIME_OF_DAY, 392
DCC, 25
Block, 294
DCCAux Task, 299
DCCAux_2 Task, 299
T1(DCC) task, 296
T2(DCC) task, 297
T3(DCC) task, 298
Diagnostics
For command processing, 138
DINT#MAX, 606
DINT#MIN, 606
DINT_TO_BYTE, 389
DINT_TO_DWORD, 389
DINT_TO_INT, 389
DINT_TO_LREAL, 389
DINT_TO_REAL, 389
DINT_TO_SINT, 389
DINT_TO_STRING, 395
DINT_TO_UDINT, 389
DINT_TO_UINT, 389
DINT_TO_USINT, 389
DINT_TO_WORD, 389
DINT_VALUE_TO_BOOL, 389
Down counter (system FB), 532
Download
CPU / drive unit, 562
Project, 555
To memory card or hard disk, 583
Download in RUN
Changed sources, 568
Changed technology objects, 580
Without HW Config, 579
Draft program
Useful notes, 601
Drive axis, 130
DriveAxis, 130

DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE, 359, 362
DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED, 359, 362
DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID, 358, 361, 363
DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR, 359, 362
DSC_SVS_DEVICE_ALARMS_IV_CALL, 359, 362
DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL, 359, 362
DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP, 359, 362
DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED, 358, 361
DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED, 359, 362
DSC_SVS_DEVICE_ALARMS_NO_ENTRY, 359, 362
DSC_SVS_DEVICE_INPUT, 170
DSC_SVS_DEVICE_OUTPUT, 170
DT_TO_DATE, 392
DT_TO_TOD, 392
DWORD_TO_BOOL, 389
DWORD_TO_BYTE, 389
DWORD_TO_DINT, 389
DWORD_TO_INT, 389
DWORD_TO_REAL, 389
DWORD_TO_SINT, 389
DWORD_TO_UDINT, 389
DWORD_TO_UINT, 389
DWORD_TO_USINT, 389
DWORD_TO_WORD, 389
DWORD_VALUE_TO_LREAL, 389
DWORD_VALUE_TO_REAL, 389

E

Edge detection
  In message programming, 472
  System FBs, 528
effectiveTaskruntime, 262
Efficient programming, 598
encoder
  External, 130
END_SYNC
  Application, 514
ENUM_TO_DINT, 393
Enumeration data types, 128
Enumerators, 127
Error
  CommandId missing, 592
  Comparing REAL variables, 596
  CPU not in RUN, 594
  Cyclic tasks, 591
  Data type conversion, 591
  during operations with floating-point numbers, 155
  Locating, 593
  Range violation, 597
  Return values of commands, 122
  runtime, 153
  TO function in cycle, 591
  while accessing system data, 193
Error activation, 182
errorgroup, 180
Event-driven tasks, 202
Example
  Expression, 331
  Using data types of TOs, 131
  WaitForCondition, 331
Execution errors, 154
Execution level, 317
  Interrupts, 318
  Overview, 197
  Round robin, 318
  Tasks, 199
  Time-controlled, 318
Execution system, 197
  configuring, 244
  Symbol browser, 210
ExecutionFaultTask, 157, 234, 318
  TaskStartInfo, 165
EXP, 374
EXPD, 374
Expert list
  Comparing, 53, 56
  Using, 50
Expert List, 47
Exponentiation, 374
EXPRESSION
  Description (in context), 329
EXPT, 374
External encoder, 45, 130
ExternalEncoderType, 130

F

F_TRIG, 529
Falling edge
  System FB, 529
Fault, 153
  Cyclic tasks, 592
  Wait times in cycle, 592
FBD, 24
Fields of application, 27
Filter for error numbers, 477

Fixed gear, 46
Floating-point number
Error cause, 596
Floating-point numbers
Error, 155
Following axis, 130
Following object, 45, 130
FollowingAxis, 130
FollowingObjectType, 130
Formula object, 46
FPU exception, 155
Free-running tasks, 201
FROM_BIG_ENDIAN
Description, 402
FROM_LITTLE_ENDIAN
Description, 402
Function
Communication, 504
Error sources during a call, 591, 592
Semaphores (application), 478
Semaphores (description);, 417
Standard function, 373
Function block
Efficient parameter access, 599
System function block, 523
Function chart, 24
Function parameters
System functions, 126

G

General interconnection screen form, 43
General standard functions, 374
Global control telegram (GC), 284
Global response
Technological alarms, 181

H

Hardware platforms, 29
HMI (Human Machine Interface), 519
HW Config, 22

I

I/O processing
Isochronous, 283
I/O variables
Assign drive parameters, 96
Assigning, 100
Identifier
Reserved for basic system, 607
Implicit interconnection, 42
Infinity, 155
Initialization
Data for STOP-RUN transition, 576
Source and TO data separated, 567
Technology objects, 130
InputSynchronousTask_1
TaskStartInfo, 164
InputSynchronousTask_1:, 225, 318
InputSynchronousTask_2, 225, 318
TaskStartInfo, 164
Instantiation, 34, 36
INT#MAX, 606
INT#MIN, 606
INT_TO_BYTE, 389
INT_TO_DINT, 389
INT_TO_DWORD, 389
INT_TO_LREAL, 389
INT_TO_REAL, 389
INT_TO_SINT, 389
INT_TO_TIME, 392
INT_TO_UDINT, 389
INT_TO_UINT, 389
INT_TO_USINT, 389
INT_TO_WORD, 390
INT_VALUE_TO_BOOL, 390
Interconnection
implicit, 42
of technology objects, 56
Via general interconnection screen forms, 43
via technological interconnection screen forms, 42
Interconnection overview, 57
Interconnection table, 58
Interface arrangements, 504
Interpolator cycle clock, 250
Interpolator cycle clock 2, 250
Interrupt
Programmable, 329
IPO cycle clock, 198, 250
IPO_2 cycle clock, 250
IPO_fast, 198, 250, 264
IPO2 - cycle clock 2, 198
IPOsynchronousTask, 228, 318
TaskStartInfo, 164
IPOsynchronousTask_2, 228, 318
TaskStartInfo, 164
IPOsynchronousTask_fast
TaskStartInfo, 164
IPOTask, 228

IPOTask_2, 228
IS_VALID
    Description, 410
Isochronous data processing, 285
Isochronous I/O processing, 283

L

Ladder diagram, 24
Level overflow, 260
    Monitoring, 262
Library
    Technology package, 152
License
    Protection against deletion, 590
LIMIT, 416
LittleByteArray_to_AnyType
    Description, 398
LN, 375
Loading
    CPU / drive unit, 562
    Data from the hard disk to the card, 584
    Project to PG, 586
    To file system, 583
Local response
    Technological alarms, 181
LOG, 375
Logarithmic standard functions, 374
logDiagAdrIoType, 170
LOWER_BOUND, 461
LREAL_TO_DINT, 390
LREAL_TO_INT, 390
LREAL_TO_REAL, 390
LREAL_TO_SINT, 390
LREAL_TO_STRING, 395
LREAL_TO_UDINT, 390
LREAL_TO_UINT, 390
LREAL_TO_USINT, 390
LREAL_VALUE_TO_BOOL, 390
LREAL_VALUE_TO_BYTE, 390
LREAL_VALUE_TO_DWORD, 390
LREAL_VALUE_TO_WORD, 390

M

MACF, 266
Marshalling, 396
Masked error numbers, 477
MAX, 414
Measuring input, 45, 130
MeasuringInputType, 130
Mechatronics, 26
Messages
    Alarms, 153
    Programming, 356, 471
MIN, 415
Monitoring
    Cycle time, 263
    Timeouts and level overflows, 262
Motion
    Interface type, 65
Motion basis, 65
Motion control, 198
Motion Control, 26, 198
Motion Control Chart, 24
MotionTasks, 201, 215
    Controlling, 578
    TaskStartInfo, 162
Multitasking
    Error sources, 598
MUX, 413

N

NaN
    Quiet, 155
    Signaling, 155
NaNq, 155
NaNs, 155
Numeric standard functions, 373

O

OFF delay (system FB), 537
Offline mode, 24
ON delay (system FB), 536
Online mode, 24
Only create program instance data once, 322
OperationLevels, 203
Output cam, 45, 130
OutputCamType, 130

P

Parameter
    Efficient access in function blocks, 599
Path axis, 130
Path object, 45, 130
PeripheralFaultTask, 234, 318
    TaskStartInfo, 167
Permit language extensions, 324
PLC functionality, 26

PLCopen, 24
PosAxis, 130
Position axis, 130
Position control cycle clock, 198, 250
PostControlTask_1, 225, 318
    TaskStartInfo, 164
PostControlTask_2, 225, 318
    TaskStartInfo, 164
Priorities, 207
Process alarms, 174
Program
    Assigning tasks, 320
    Increasing efficiency, 598
    Locating errors, 593, 594
Programming, 24
    Command execution; command execution;, 131
    Error sources, 591
    Increasing efficiency, 598
Programs, 197
    Assigning tasks, 244
Project, 23
    Check consistency, 559
    compilation, 559
    Download, 555
ProTool, 519
Pulse (system FB), 536
Pulse-width modulation cycle clock, 251
PWMsynchronousTask, 225, 318
    TaskStartInfo, 164

R

R_TRIG, 528
Range violation, 597
REAL_TO_DINT, 390
REAL_TO_DWORD, 390
REAL_TO_INT, 390
REAL_TO_LREAL, 390
REAL_TO_SINT, 390
REAL_TO_STRING, 395
REAL_TO_TIME, 392
REAL_TO_UDINT, 390
REAL_TO_UINT, 390
REAL_TO_USINT, 390
REAL_VALUE_TO_BOOL, 390
REAL_VALUE_TO_BYTE, 390
REAL_VALUE_TO_DWORD, 390
REAL_VALUE_TO_WORD, 390
Real-time clock
    Resetting the time, 314
    Set time, 311
    Synchronizing the SINAMICS clock, 313
    System FB, 538
References, 4
Reserved identifiers, 607
Return value, 37, 192
Rising edge
    System FB, 528
ROL, 376
ROR, 376
Round robin, 201, 273
    Setting time allocation, 276
RTC, 538
RUN
    Effect on variable initialization, 321
    Mode not reached, 594
Runtime group, 294
Runtime model, 209
Runtime system, 200

S

SEL, 412
Semaphore commands
    Application), 478
    Description, 417
Sending
    Data, 504
Sensor, 46
Sequence
    in the round robin execution level, 274
Sequential program execution
    Determining, 320
    Influencing, 338
    Status of the task, 327
Sequential tasks, 202
Servo cycle clock, 250
Servo_fast, 198, 250, 264
ServoSynchronousTask, 226, 318
    TaskStartInfo, 164
ServoSynchronousTask_fast
    TaskStartInfo, 164
Set flip-flop, 525
Setting up address, 110
SHL, 376
SHR, 377
ShutdownTask, 202, 242
    TaskStartInfo, 177
SIMATIC S7 device
    Communication with SIMOTION devices, 508
    Destination address, structure, 505
SIMOTION
    Execution system, 197

Fields of application, 27
Motion control, 26
Project, 23
Runtime model, 209
Runtime system, 200
System architecture, 21
Technology packages, 34
SIMOTION C2xx, 30
SIMOTION D4xx, 30
SIMOTION device communication with SIMATIC S7 device, 508
SIMOTION P320, 30
SIMOTION P350, 30
SIMOTION SCOUT engineering system, 22
SIN, 375
SINT#MAX, 606
SINT#MIN, 606
SINT_TO_BYTE, 390
SINT_TO_DINT, 390
SINT_TO_DWORD, 390
SINT_TO_INT, 390
SINT_TO_LREAL, 390
SINT_TO_REAL, 390
SINT_TO_UDINT, 390
SINT_TO_UINT, 390
SINT_TO_USINT, 390
SINT_TO_WORD, 390
SINT_VALUE_TO_BOOL, 390
SQRT, 374
SR, 526, 527
Stack size, 249
Standard functions, 373
Start sequence
Tasks, 208
StartupTask, 201, 212
TaskStartInfo, 161
Status
Task (status values), 327
STOP mode, 153
STOP to RUN
Effect on variable initialization, 321
Error cause, 594
STRING
Processing functions, 384
STRING_TO_DINT, 395
STRING_TO_LREAL, 395
STRING_TO_REAL, 395
STRING_TO_UDINT, 395
STRUCTALARMID#NIL, 607
StructAlarmId_TO_DINT, 390
STRUCTTASKID#NIL, 607
Structure-changing variables, 582
Structured Text, 24
Symbol browser
Execution system, 210
Symbolic assignment, 87
Activating, 116
Assignment destination, 708
Assignment dialog, 105
Deactivating, 116
External encoder, 92
I/O variables, 96
of axis and drive, 89
Setting up address, 110
TO cam track, 93
TO measuring input, 93
TO output cam, 93
TO sensor, 94
Synchronous command execution, 37
For data transmission, 504
Synchronous start, 514
Synchronous tasks, 201
SynchronousTasks, 202, 224
TaskStartInfo, 163
System architecture, 21
System clocks
Assigning, 257
Defining, 250
System cycle clock
Error cause, 594
System data
Error while accessing, 193
System function
Definition, 121
Query the command/execution status; system function: Return values;, 138
System function blocks
Definitions, 523
Overview, 523
System tasks, 197
System variables, 37
Definition, 121
Error cause, 596
Optimizing access, 599
Overview; variables: System; variables: structured; structured variables; multielement variables, 143
SystemInterruptTasks, 202, 232

T

T#MAX, 607
T#MIN, 607
TAN, 375

Task
Assigning initial values, 321
Assigning programs, 320
BackgroundTask, 218, 319
Concept, 317
Control commands (brief description), 338
ExecutionFaultTask, 234
Initialization of local variables, 321
IPOsynchronousTask, 228
MotionTask, 215, 319
PeripheralFaultTask, 234
Priorities, 205
Programming, 338
Runtime measurement (functions), 349
ServoSynchronousTask, 226
ShutdownTask, 242, 319
Start info, 178
StartupTask, 212, 319
Status, 327
SynchronousTask, 318
SynchronousTasks, 224
SystemInterruptTask, 318
SystemInterruptTasks, 232
TaskStartInfo (TSI), 263
TechnologicalFaultTask, 234
TimeFaultBackgroundTask, 234
TimeFaultTask, 233
TimerInterruptTask, 221, 318
UserInterruptTask, 318
UserInterruptTasks, 237
Task control, 249
Task priorities, 205
Task runtimes, 258
Task Trace, 283
TASK_STATE_INVALID, 327, 341
TASK_STATE_LOCKED, 342
TASK_STATE_RUNNING, 327, 341
TASK_STATE_STOP_PENDING, 327, 341
TASK_STATE_STOPPED, 327, 341
TASK_STATE_SUSPENDED, 327, 341
TASK_STATE_WAIT_NEXT_CYCLE, 327, 341
TASK_STATE_WAIT_NEXT_INTERRUPT, 327, 342
TASK_STATE_WAITING, 327, 341
Taskruntime, 262
Tasks, 197, 199
Assigning programs, 244
Start sequence, 208
TaskStartInfo, 178
TaskStartInfo (TSI), 263
TControl
Technology package, 35
Tdp, 286
Tdx, 286
Technological alarm, 40
Technological alarms, 179
configuring, 183
Evaluating in the user program, 191
Global response, 181
Local response, 181
Technological interconnection screen forms, 42
TechnologicalFaultTask, 234, 318
TaskStartInfo, 175
Technology object, 32, 36
Configuration, 36
Configuration data; configuration data., 147
Data types, 130
Definition, 121
Functions (codes), 122
Functions (definition), 121
Functions (input parameters), 126
Initialization, 130
Instances, 122
Instantiation, 34, 36
Interconnection, 56
Names, 122
Package (definition), 121
Querying TaskStartInfo, 178
reset, 151
System variables, 37
Validity check, 130
Technology object trace, 67
Technology object types, 34, 36
Technology package
Definition, 121
in library, 152
Technology packages, 34
Temperature channel, 45
Temperature control, 45, 199
Temperature controller, 130
TemperatureControllerType, 130
Terminal - axis - response time, 225
Terminal - terminal response time, 225
Terminal-terminal isochronous mode, 292
Ti, 286
Time allocation
in the round robin execution level, 273
Setting, 276
Time types
Conversions, 392
TIME#MAX, 607
TIME#MIN, 607
TIME_OF_DAY#MAX, 607
TIME_OF_DAY#MIN, 607
TIME_TO_INT, 392

TIME_TO_REAL, 392
TIME_TO_UDINT, 392
TimeFaultBackgroundTask, 234, 318
    TaskStartInfo, 176
TimeFaultTask, 233, 318
    TaskStartInfo, 177
Timeout, 260, 592
    Monitoring, 262
TimerInterruptTask, 221
TimerInterruptTasks, 202
    TaskStartInfo, 162
Time-triggered tasks, 201
To, 286
TO, 32
TO axis
    Assigning I/O variables telegram, 95
TO cam track
    Assigning, 93
TO external encoder
    Assigning, 92
TO measuring input
    Assigning, 93
TO output cam
    Assigning, 93
TO trace, 67
TO#NIL, 130, 145, 607
TO_BIG_ENDIAN
    Description, 401
TO_LITTLE_ENDIAN
    Description, 401
TOD#MAX, 607
TOD#MIN, 607
TOF, 537
TON, 536
Tooltips, 37
Torque limits B+ / B, 66
Totally Integrated Automation, 28
TP, 536
TP Cam
    Technology package, 34
TP Cam_ext
    Technology package, 34
TP Path
    Path object, 34
Trigonometric standard functions, 375
TRUNC, 374
TSI, 263
TSI#Alarmgroup, 176
TSI#alarmNumber, 175
TSI#alarmP1_DINT, 175
TSI#alarmP1_LREAL, 175
TSI#alarmP1_UDINT, 175
TSI#alarmP2_DINT, 175
TSI#alarmP2_LREAL, 175
TSI#alarmP2_UDINT, 175
TSI#alarmP3_DINT, 175
TSI#alarmP3_LREAL, 175
TSI#alarmP3_UDINT, 175
TSI#alarmP4_DINT, 175
TSI#alarmP4_LREAL, 175
TSI#alarmP4_UDINT, 175
TSI#alarmP5_DINT, 175
TSI#alarmP5_LREAL, 175
TSI#alarmP5_UDINT, 175
TSI#AlarmType, 175
TSI#commandId.high, 175
TSI#commandId.low, 175
TSI#currentTaskId, 161, 162, 163, 165, 167, 175, 176, 177
TSI#cycleTime, 161, 162, 163, 164, 165, 167, 175, 176, 177
TSI#cycleTime_us, 161, 162, 163, 164, 165, 167, 175, 176, 177
TSI#details, 170, 171
TSI#dwuser_1, 161, 162, 163, 164, 165, 167, 175, 177
TSI#dwuser_2, 161, 162, 163, 164, 165, 167, 175, 177, 178
TSI#eventClass, 170, 172
TSI#executionFaultType, 166
TSI#faultId, 171, 172
TSI#interruptID, 167, 171, 172, 177
TSI#localReaction:, 176
TSI#logBaseAdrIn, 170
TSI#logBaseAdrOut, 170
TSI#logDiagAdr, 170
TSI#shutDownInitiator, 178
TSI#startTime, 161, 162, 163, 165, 167, 175, 176, 177
TSI#taskId, 166, 177
TSI#taskType, 161, 162, 163, 164, 166, 171, 176, 177, 178
TSI#toInst, 175
Type conversion functions, 387

U

UDINT#MAX, 607
UDINT#MIN, 607
UDINT_TO_BYTE, 390
UDINT_TO_DINT, 390
UDINT_TO_DWORD, 390
UDINT_TO_INT, 390
UDINT_TO_LREAL, 391

UDINT_TO_REAL, 391
UDINT_TO_SINT, 391
UDINT_TO_STRING, 395
UDINT_TO_TIME, 392
UDINT_TO_UINT, 391
UDINT_TO_USINT, 391
UDINT_TO_WORD, 391
UDINT_VALUE_TO_BOOL, 391
UINT#MAX, 607
UINT#MIN, 607
UINT_TO_BYTE, 391
UINT_TO_DINT, 391
UINT_TO_DWORD, 391
UINT_TO_INT, 391
UINT_TO_LREAL, 391
UINT_TO_REAL, 391
UINT_TO_SINT, 391
UINT_TO_UDINT, 391
UINT_TO_USINT, 391
UINT_TO_WORD, 391
UINT_VALUE_TO_BOOL, 391
Up counter (system FB), 530
Up/down counter (system FB), 534
UPPER_BOUND, 462
USEPACKAGE, 122
User data
    Deleting, 590
User interrupt
    Programmable, 329
User program
    Overview, 24
    Tasks, 199
UserInterruptTasks, 202, 237
    TaskStartInfo, 163
USINT#MAX, 606
USINT#MIN, 606
USINT_TO_BYTE, 391
USINT_TO_DINT, 391
USINT_TO_DWORD, 391
USINT_TO_INT, 391
USINT_TO_LREAL, 391
USINT_TO_REAL, 391
USINT_TO_SINT, 391
USINT_TO_UDINT, 391
USINT_TO_UINT, 391
USINT_TO_WORD, 391
USINT_VALUE_TO_BOOL, 391

V

Value specification; reference parameter; parameter
    Value specification; parameter: Reference parameter, 129
Variable initialization
    in program sources, 576
Variables
    Efficient access, 598
    Instance declaration of TO, 122
    Range violation, 597
    Semaphores (application), 478

W

Wait for condition, 207
WAITFORCONDITION, 207
    Description (in context), 329
Watchdog, 263
WORD_TO_BOOL, 391
WORD_TO_BYTE, 391
WORD_TO_DINT, 391
WORD_TO_DWORD, 391
WORD_TO_INT, 391
WORD_TO_SINT, 391
WORD_TO_UDINT, 391
WORD_TO_UINT, 391
WORD_TO_USINT, 391
WORD_VALUE_TO_LREAL, 391
WORD_VALUE_TO_REAL, 391