SIEMENS

SIMATIC

S7
S7-1200 Programmable controller

System Manual

Preface

Product overview 1

STEP 7 programming software 2

Installation 3

PLC concepts 4

Device configuration 5

Programming concepts 6

Basic instructions 7

Extended instructions 8

Technology instructions 9

Communication 10

Web server 11

Communication processor 12

Teleservice communication (SMTP e-mail) 13

Online and diagnostic tools 14

Technical specifications A

Calculating a power budget B

Order numbers C

11/2011
A5E02486680-05

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

with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

CAUTION

without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

NOTICE

indicates that an unintended result or situation can occur if the relevant information is not taken into account.

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 AG. 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 AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
GERMANY

Order number: 6ES7298-8FA30-8BH0
℗ 11/2011

Copyright © Siemens AG 2011.
Technical data subject to change

Preface

Purpose of the manual

The S7-1200 series is a line of programmable logic controllers (PLCs) that can control a variety of automation applications. Compact design, low cost, and a powerful instruction set make the S7-1200 a perfect solution for controlling a wide variety of applications. The S7-1200 models and the Windows-based programming tool give you the flexibility you need to solve your automation problems.

This manual provides information about installing and programming the S7-1200 PLCs and is designed for engineers, programmers, installers, and electricians who have a general knowledge of programmable logic controllers.

Required basic knowledge

To understand this manual, it is necessary to have a general knowledge of automation and programmable logic controllers.

Scope of the manual

This manual describes the following products:

- STEP 7 V11 Basic and Professional
- S7-1200 CPU firmware release V2.2

For a complete list of the S7-1200 products described in this manual, refer to the technical specifications (Page 657).

Certification, CE label, C-Tick, and other standards

Refer to the technical specifications (Page 657) for more information.

Service and support

In addition to our documentation, we offer our technical expertise on the Internet on the customer support web site (http://www.siemens.com/automation/support-request).

Contact your Siemens distributor or sales office for assistance in answering any technical questions, for training, or for ordering S7 products. Because your sales representatives are technically trained and have the most specific knowledge about your operations, process and industry, as well as about the individual Siemens products that you are using, they can provide the fastest and most efficient answers to any problems you might encounter.

Documentation and information

S7-1200 and STEP 7 provide a variety of documentation and other resources for finding the technical information that you require.

- The S7-1200 system manual provides specific information about the operation, programming and the specifications for the complete S7-1200 product family. In addition to the system manual, the S7-1200 Easy Book provides a more general overview to the capabilities of the S7-1200 family.

  Both the system manual and the Easy Book are available as electronic (PDF) and printed manuals. The electronic manuals can be downloaded from the customer support web site and can also be found on the companion disk that ships with every S7-1200 CPU.

- The online information system of STEP 7 provides immediate access to the conceptual information and specific instructions that describe the operation and functionality of the programming package and basic operation of SIMATIC CPUs.

- My Documentation Manager accesses the electronic (PDF) versions of the SIMATIC documentation set, including the system manual, the Easy Book and the information system of STEP 7. With My Documentation Manager, you can drag and drop topics from various documents to create your own custom manual.

  The customer support entry portal (http://support.automation.siemens.com) provides a link to My Documentation Manager under mySupport.

- The customer support web site also provides podcasts, FAQs, and other helpful documents for S7-1200 and STEP 7. The podcasts utilize short educational video presentations that focus on specific features or scenarios in order to demonstrate the interactions, convenience and efficiency provided by STEP 7. Visit the following web sites to access the collection of podcasts:

  – STEP 7 Basic web page (http://www.automation.siemens.com/mcms/simatic-controller-software/en/step7/step7-basic/Pages/Default.aspx)

  – STEP 7 Professional web page (http://www.automation.siemens.com/mcms/simatic-controller-software/en/step7/step7-professional/Pages/Default.aspx)

- You can also follow or join product discussions on the Service & Support technical forum (https://www.automation.siemens.com/WW/forum/guests/Conferences.aspx?Language=en&siteid=csius&treeLang=en&groupid=4000002&extranet=standard&viewreg=WW&nodeid0=34612486). These forums allow you to interact with various product experts.

  – Forum for S7-1200 (https://www.automation.siemens.com/WW/forum/guests/Conference.aspx?SortField=LastPostDate&SortOrder=Descending&ForumID=258&Language=en&onlyInternet=False)

  – Forum for STEP 7 Basic (https://www.automation.siemens.com/WW/forum/guests/Conference.aspx?SortField=LastPostDate&SortOrder=Descending&ForumID=265&Language=en&onlyInternet=False)

Table of contents

Preface **3**

1 Product overview **17**

1.1 Introducing the S7-1200 PLC 17

1.2 Expansion capability of the CPU 20

1.3 S7-1200 modules 22

1.4 New features for S7-1200 and STEP 7 V11 23

1.5 Basic HMI panels 25

2 STEP 7 programming software **27**

2.1 System requirements 27

2.2 Different views to make the work easier 28

2.3 Easy-to-use tools 29
2.3.1 Inserting instructions into your user program 29
2.3.2 Accessing instructions from the "Favorites" toolbar 29
2.3.3 Creating a complex equation with a simple instruction 30
2.3.4 Adding inputs or outputs to a LAD or FBD instruction 31
2.3.5 Expandable instructions 32
2.3.6 Selecting a version for an instruction 32
2.3.7 Modifying the appearance and configuration of STEP 7 33
2.3.8 Dragging and dropping between editors 33
2.3.9 Changing the operating mode of the CPU 34
2.3.10 Capturing and restoring a block state 34
2.3.11 Changing the call type for a DB 36
2.3.12 Temporarily disconnecting devices from a network 37
2.3.13 Virtual unplugging of devices from the configuration 38

3 Installation **39**

3.1 Guidelines for installing S7-1200 devices 39

3.2 Power budget 40

3.3 Installation and removal procedures 42
3.3.1 Mounting dimensions for the S7-1200 devices 42
3.3.2 Installing and removing the CPU 44
3.3.3 Installing and removing an SB or a CB 46
3.3.4 Installing and removing an SM 47
3.3.5 Installing and removing a CM or CP 48
3.3.6 Removing and reinstalling the S7-1200 terminal block connector 50
3.3.7 Installing and removing the expansion cable 51
3.3.8 TS (teleservice) adapter 52
3.3.8.1 Connecting the TeleService Adapter 52
3.3.8.2 Installing the SIM card 53
3.3.8.3 Installing the TS adapter unit 54

3.3.8.4 Installing the TS adapter on a wall ... 56
3.4 Wiring guidelines ... 56
4 PLC concepts ... 61
4.1 Execution of the user program ... 61
4.1.1 Operating modes of the CPU ... 63
4.1.2 Processing the scan cycle in RUN mode ... 66
4.1.3 Organization blocks (OBs) ... 66
4.1.4 Event execution priorities and queuing ... 68
4.1.5 Monitoring the cycle time ... 73
4.1.6 CPU memory ... 75
4.1.6.1 System and clock memory ... 78
4.1.6.2 Configuring the outputs on a RUN-to-STOP transition ... 80
4.2 Data storage, memory areas, I/O and addressing ... 80
4.2.1 Accessing the data of the S7-1200 ... 80
4.3 Processing of analog values ... 85
4.4 Data types ... 86
4.4.1 Bool, Byte, Word, and DWord data types ... 87
4.4.2 Integer data types ... 88
4.4.3 Floating-point real data types ... 88
4.4.4 Time and Date data types ... 89
4.4.5 Data structure data type ... 92
4.4.6 PLC data type ... 93
4.4.7 Pointer data types ... 93
4.4.7.1 "Pointer" pointer data type ... 94
4.4.7.2 "Any" pointer data type ... 95
4.4.7.3 "Variant" pointer data type ... 96
4.4.8 Accessing a "slice" of a tagged data type ... 96
4.4.9 Accessing a tag with an AT overlay ... 98
4.5 Using a memory card ... 99
4.5.1 Inserting a memory card in the CPU ... 100
4.5.2 Configuring the startup parameter of the CPU before copying the project to the memory card ... 102
4.5.3 Transfer card ... 102
4.5.4 Program card ... 104
4.6 Recovery from a lost password ... 107
5 Device configuration ... 109
5.1 Inserting a CPU ... 110
5.2 Detecting the configuration for an unspecified CPU ... 111
5.3 Adding modules to the configuration ... 112
5.4 Configuring the operation of the CPU ... 114
5.5 Configuring the parameters of the modules ... 115
5.6 Configuring the CPU for communication ... 117
5.6.1 Creating a network connection ... 117
5.6.2 Configuring the Local/Partner connection path ... 118
5.6.3 Parameters for the PROFINET connection ... 120

5.6.4 Assigning Internet Protocol (IP) addresses 123
5.6.4.1 Assigning IP addresses to programming and network devices 123
5.6.4.2 Checking the IP address of your programming device 125
5.6.4.3 Assigning an IP address to a CPU online 125
5.6.4.4 Configuring an IP address for a CPU in your project 127
5.6.5 Testing the PROFINET network 129
5.6.6 Locating the Ethernet (MAC) address on the CPU 130
5.6.7 Configuring Network Time Protocol synchronization 131
5.6.8 PROFINET device start-up time, naming, and address assignment 132

6 Programming concepts 135

6.1 Guidelines for designing a PLC system 135

6.2 Structuring your user program 136

6.3 Using blocks to structure your program 138
6.3.1 Organization block (OB) 138
6.3.2 Function (FC) 140
6.3.3 Function block (FB) 140
6.3.4 Data block (DB) 141

6.4 Understanding data consistency 143

6.5 Programming language 144
6.5.1 Ladder logic (LAD) 144
6.5.2 Function Block Diagram (FBD) 145
6.5.3 SCL 145
6.5.4 EN and ENO for LAD, FBD and SCL 152

6.6 Protection 153
6.6.1 Access protection for the CPU 153
6.6.2 Know-how protection 154
6.6.3 Copy protection 156

6.7 Downloading the elements of your program 157

6.8 Uploading from the CPU 157
6.8.1 Copying elements of the project 157
6.8.2 Using the compare function 158

6.9 Debugging and testing the program 159
6.9.1 Monitor and modify data in the CPU 159
6.9.2 Watch tables and force tables 159
6.9.3 Cross reference to show usage 160
6.9.4 Call structure to examine the calling hierarchy 161

7 Basic instructions 163

7.1 Bit logic 163
7.1.1 Bit logic contacts and coils 163
7.1.2 Set and reset instructions 166
7.1.3 Positive and negative edge instructions 168

7.2 Timers 170

7.3 Counters 178

7.4 Compare 183
7.4.1 Compare 183

7.4.2 In-range and Out-of-range instructions ... 184
7.4.3 OK and Not OK instructions ... 185

7.5 Math ... 186
7.5.1 Calculate instruction ... 186
7.5.2 Add, subtract, multiply and divide instructions ... 187
7.5.3 Modulo instruction ... 188
7.5.4 Negation instruction ... 189
7.5.5 Increment and decrement instructions ... 189
7.5.6 Absolute value instruction ... 190
7.5.7 Minimum and Maximum instructions ... 191
7.5.8 Limit instruction ... 192
7.5.9 Floating-point math instructions ... 192

7.6 Move ... 194
7.6.1 Move and block move instructions ... 194
7.6.2 FieldRead and FieldWrite instructions ... 197
7.6.3 Fill instructions ... 199
7.6.4 Swap instruction ... 200

7.7 Convert ... 201
7.7.1 CONV instruction ... 201
7.7.2 Conversion instructions for SCL ... 202
7.7.3 Round and truncate instructions ... 204
7.7.4 Ceiling and floor instructions ... 205
7.7.5 Scale and normalize instructions ... 206

7.8 Program control ... 209
7.8.1 Overview of SCL program control statements ... 209
7.8.2 IF-THEN statement ... 210
7.8.3 CASE statement ... 211
7.8.4 FOR statement ... 212
7.8.5 WHILE-DO statement ... 213
7.8.6 REPEAT-UNTIL statement ... 214
7.8.7 CONTINUE statement ... 214
7.8.8 EXIT statement ... 215
7.8.9 GOTO statement ... 216
7.8.10 RETURN statement ... 216
7.8.11 Jump and label instructions ... 217
7.8.12 JMP_LIST instruction ... 217
7.8.13 SWITCH instruction ... 218
7.8.14 RET execution control instruction ... 220
7.8.15 Re-trigger scan cycle watchdog instruction ... 221
7.8.16 Stop scan cycle instruction ... 222
7.8.17 Get Error instructions ... 222

7.9 Word logic operations ... 226
7.9.1 AND, OR, and XOR instructions ... 226
7.9.2 Invert instruction ... 227
7.9.3 Encode and decode instructions ... 227
7.9.4 Select, Multiplex, and Demultiplex instructions ... 228

7.10 Shift and Rotate ... 231
7.10.1 Shift instructions ... 231
7.10.2 Rotate instructions ... 232

8 Extended instructions 233

8.1 Date and time-of-day 233
8.1.1 Date and time instructions 233
8.1.2 Set and read system clock 235
8.1.3 Run-time meter instruction 237
8.1.4 SET_TIMEZONE instruction 238

8.2 String and character 240
8.2.1 String data overview 240
8.2.2 S_MOVE instruction 240
8.2.3 String conversion instructions 241
8.2.3.1 String to value and value to string conversions 241
8.2.3.2 String-to-characters and characters-to-string conversions 249
8.2.3.3 ASCII to Hex and Hex to ASCII conversions 251
8.2.4 String operation instructions 253
8.2.4.1 LEN 253
8.2.4.2 CONCAT 254
8.2.4.3 LEFT, RIGHT, and MID 255
8.2.4.4 DELETE 256
8.2.4.5 INSERT 257
8.2.4.6 REPLACE 258
8.2.4.7 FIND 259

8.3 Distributed I/O (PROFINET, PROFIBUS, or AS-i) 260
8.3.1 RDREC and WRREC 260
8.3.2 RALRM 263
8.3.3 STATUS parameter for RDREC, WRREC, and RALRM 265
8.3.4 DPRD_DAT and DPWR_DAT 269
8.3.5 DPNRM_DG 271

8.4 Interrupts 273
8.4.1 Attach and detach instructions 273
8.4.2 Cyclic interrupts 276
8.4.2.1 SET_CINT (Set cyclic interrupt) 276
8.4.2.2 QRY_CINT (Query cyclic interrupt) 278
8.4.3 Time delay interrupts 279
8.4.4 Asynchronous event interrupts 281

8.5 Diagnostics (PROFINET or PROFIBUS) 282
8.5.1 LED instruction 282
8.5.2 DeviceStates instruction 283
8.5.3 ModuleStates instruction 284
8.5.4 GET_DIAG instruction 285

8.6 Pulse 287
8.6.1 CTRL_PWM instruction 287
8.6.2 Operation of the pulse outputs 289
8.6.3 Configuring a pulse channel for PWM 290

8.7 Data logging 291
8.7.1 Data log record structure 291
8.7.2 Program instructions that control Data logs 293
8.7.2.1 DataLogCreate 293
8.7.2.2 DataLogOpen 296
8.7.2.3 DataLogClose 297

8.7.2.4 DataLogWrite ... 298
8.7.2.5 DataLogNewFile ... 300
8.7.3 Working with data logs ... 302
8.7.4 Limits to the size of data log files ... 303
8.7.5 Data log example program ... 305

8.8 Data block control ... 310
8.8.1 READ_DBL, WRIT_DBL (Read from or write to a DB in load memory) ... 310

8.9 Common error codes for the "Extended" instructions ... 313

9 Technology instructions ... 315

9.1 High-speed counter ... 315
9.1.1 Operation of the high-speed counter ... 317
9.1.2 Configuration of the HSC ... 323

9.2 PID control ... 324
9.2.1 Inserting the PID instruction and technological object ... 325
9.2.2 PID_Compact instruction ... 328
9.2.3 PID_3STEP instruction ... 331
9.2.4 Configuring the PID controller ... 337
9.2.5 Commissioning the PID controller ... 340

9.3 Basic motion control ... 340
9.3.1 Configuration of the axis ... 343
9.3.2 Motion control instructions ... 346
9.3.2.1 MC_Power instruction ... 346
9.3.2.2 MC_Reset instruction ... 349
9.3.2.3 MC_Home instruction ... 350
9.3.2.4 MC_Halt instruction ... 352
9.3.2.5 MC_MoveAbsolute instruction ... 354
9.3.2.6 MC_MoveRelative instruction ... 356
9.3.2.7 MC_MoveVelocity instruction ... 358
9.3.2.8 MC_MoveJog instruction ... 360
9.3.2.9 MC_CommandTable instruction ... 362
9.3.2.10 MC_ChangeDynamic ... 365
9.3.3 Operation of motion control for S7-1200 ... 367
9.3.3.1 CPU outputs used for motion control ... 367
9.3.3.2 Hardware and software limit switches for motion control ... 368
9.3.3.3 Homing ... 371
9.3.3.4 Jerk limit ... 376
9.3.4 Commissioning ... 377
9.3.5 Monitoring active commands ... 380
9.3.5.1 Monitoring MC instructions with a "Done" output parameter ... 380
9.3.5.2 Monitoring the MC_Velocity instruction ... 384
9.3.5.3 Monitoring the MC_MoveJog instruction ... 388

10 Communication ... 393

10.1 Number of asynchronous communication connections supported ... 394

10.2 PROFINET ... 394
10.2.1 Local/Partner connection ... 394
10.2.2 Open user communication ... 396
10.2.2.1 Connection IDs for the PROFINET instructions ... 396
10.2.2.2 Protocols ... 399

10.2.2.3 Ad hoc mode 400
10.2.2.4 TCP and ISO on TCP 401
10.2.2.5 UDP 415
10.2.2.6 T_CONFIG 421
10.2.2.7 Common parameters for instructions 427
10.2.3 Communication with a programming device 429
10.2.3.1 Establishing the hardware communications connection 429
10.2.3.2 Configuring the devices 430
10.2.3.3 Assigning Internet Protocol (IP) addresses 431
10.2.3.4 Testing your PROFINET network 431
10.2.4 HMI-to-PLC communication 431
10.2.4.1 Configuring logical network connections between two devices 432
10.2.5 PLC-to-PLC communication 433
10.2.5.1 Configuring logical network connections between two devices 434
10.2.5.2 Configuring the Local/Partner connection path between two devices 434
10.2.5.3 Configuring transmit (send) and receive parameters 434
10.2.6 Configuring a CPU and PROFINET IO device 437
10.2.7 Diagnostics 440
10.2.8 Distributed I/O Instructions 441
10.2.9 Diagnostic instructions 441
10.2.10 Diagnostic events for distributed I/O 442

10.3 PROFIBUS 442
10.3.1 Communications modules PROFIBUS 444
10.3.1.1 Connecting to PROFIBUS 444
10.3.1.2 Communications services of the PROFIBUS CMs 444
10.3.1.3 Other properties of the PROFIBUS CMs 446
10.3.1.4 Configuration examples for PROFIBUS 447
10.3.2 Configuring a DP master and slave device 448
10.3.2.1 Adding the CM 1243-5 (DP master) module and a DP slave 448
10.3.2.2 Configuring logical network connections between two PROFIBUS devices 449
10.3.2.3 Assigning PROFIBUS addresses to the CM 1243-5 module and DP slave 449
10.3.3 Distributed I/O Instructions 451
10.3.4 Diagnostic instructions 451
10.3.5 Diagnostic events for distributed I/O 451

10.4 AS-i 452
10.4.1 Configuring an AS-i master and slave device 453
10.4.1.1 Adding the CM 1243-2 AS-i Master module and AS-i slave 453
10.4.1.2 Configuring logical network connections between two AS-i devices 453
10.4.1.3 Assigning AS-i addresses to the CM 1243-2 AS-i Master module and AS-i slave 454
10.4.2 Exchanging data between the user program and AS-i slaves 455
10.4.2.1 Configuring slaves with STEP 7 455
10.4.2.2 Configuring slaves without STEP 7 457
10.4.3 Distributed I/O Instructions 458
10.4.4 Working with AS-i online tools 458

10.5 S7 communication 460
10.5.1 GET and PUT instructions 460
10.5.2 Creating an S7 connection 464
10.5.3 Configuring the Local/Partner connection path between two devices 464
10.5.4 GET/PUT connection parameter assignment 465
10.5.4.1 Connection parameters 465
10.5.4.2 Configuring a CPU-to-CPU S7 connection 468

11 Web server 473

11.1 Enabling the Web server 474

11.2 Standard web pages 474
11.2.1 Accessing the standard Web pages from the PC 474
11.2.2 Layout of the standard Web pages 476
11.2.3 Introduction 478
11.2.4 Start 479
11.2.5 Identification 480
11.2.6 Diagnostic Buffer 480
11.2.7 Module Information 481
11.2.8 Communication 483
11.2.9 Variable Status 484
11.2.10 Data Logs 486
11.2.11 Constraints 488
11.2.11.1 Features restricted when JavaScript is disabled 489
11.2.11.2 Features restricted when cookies are not allowed 490
11.2.11.3 Importing the Siemens security certificate 490
11.2.11.4 Importing CSV format data logs to non-USA/UK versions of Microsoft Excel 491

11.3 User-defined web pages 492
11.3.1 Creating HTML pages 493
11.3.2 AWP commands supported by the S7-1200 Web server 494
11.3.2.1 Reading variables 495
11.3.2.2 Writing variables 496
11.3.2.3 Reading special variables 497
11.3.2.4 Writing special variables 499
11.3.2.5 Using an alias for a variable reference 500
11.3.2.6 Defining enum types 501
11.3.2.7 Referencing CPU variables with an enum type 501
11.3.2.8 Creating fragments 503
11.3.2.9 Importing fragments 504
11.3.2.10 Combining definitions 504
11.3.2.11 Handling tag names that contain special characters 505
11.3.3 Configuring use of user-defined Web pages 507
11.3.4 Programming the WWW instruction for user-defined web pages 508
11.3.5 Downloading the program blocks to the CPU 509
11.3.6 Accessing the user-defined web pages from the PC 510
11.3.7 Constraints specific to user-defined Web pages 510
11.3.8 Example of a user-defined web page 511
11.3.8.1 Web page for monitoring and controlling a wind turbine 511
11.3.8.2 Reading and displaying controller data 513
11.3.8.3 Using an enum type 514
11.3.8.4 Writing user input to the controller 515
11.3.8.5 Writing a special variable 516
11.3.8.6 Reference: HTML listing of remote wind turbine monitor Web page 516
11.3.8.7 Configuration in STEP 7 of the example Web page 520
11.3.9 Setting up user-defined Web pages in multiple languages 522
11.3.9.1 Creating the folder structure 522
11.3.9.2 Programming the language switch 523
11.3.9.3 Configuring STEP 7 to use a multi-language page structure 525
11.3.10 Advanced user-defined Web page control 526

12 Communication processor 529
12.1 Using the RS232 and RS485 communication interfaces 529
12.2 Biasing and terminating an RS485 network connector 530
12.3 Point-to-Point (PtP) communication 531
12.3.1 Point-to-Point instructions 532
12.3.1.1 Common parameters for Point-to-Point instructions 532
12.3.1.2 PORT_CFG instruction 534
12.3.1.3 SEND_CFG instruction 535
12.3.1.4 RCV_CFG instruction 537
12.3.1.5 SEND_PTP instruction 541
12.3.1.6 RCV_PTP instruction 544
12.3.1.7 RCV_RST instruction 546
12.3.1.8 SGN_GET instruction 547
12.3.1.9 SGN_SET instruction 548
12.3.2 Configuring the communication ports 549
12.3.2.1 Managing flow control 550
12.3.3 Configuring the transmit (send) and receive parameters 552
12.3.3.1 Configuring transmit (send) parameters 552
12.3.3.2 Configuring receive parameters 553
12.3.4 Programming the PtP communications 560
12.3.4.1 Polling architecture 561
12.3.5 Example: Point-to-Point communication 562
12.3.5.1 Configuring the communication module 563
12.3.5.2 Programming the STEP 7 program 565
12.3.5.3 Configuring the terminal emulator 566
12.3.5.4 Running the example program 567
12.4 Universal serial interface (USS) communication 567
12.4.1 Requirements for using the USS protocol 568
12.4.2 USS_DRV instruction 570
12.4.3 USS_PORT instruction 573
12.4.4 USS_RPM instruction 574
12.4.5 USS_WPM instruction 575
12.4.6 USS status codes 576
12.4.7 General drive setup information 578
12.5 Modbus communication 581
12.5.1 Overview of Modbus RTU and TCP communication 581
12.5.2 Modbus TCP 584
12.5.2.1 MB_CLIENT (Modbus TCP) 584
12.5.2.2 MB_SERVER (Modbus TCP) 590
12.5.2.3 MB_SERVER example: Multiple TCP connections 595
12.5.2.4 MB_CLIENT example 1: Multiple requests with common TCP connection 596
12.5.2.5 MB_CLIENT example 2: Multiple requests with different TCP connections 597
12.5.2.6 MB_CLIENT example 3: Output image write request 598
12.5.2.7 MB_CLIENT example 4: Coordinating multiple requests 598
12.5.3 Modbus RTU 599
12.5.3.1 MB_COMM_LOAD 600
12.5.3.2 MB_MASTER 603
12.5.3.3 MB_SLAVE 608
12.5.3.4 Modbus RTU master example program 615
12.5.3.5 Modbus RTU slave example program 617

12.6 Telecontrol and TeleService with the CP 1242-7 ... 618
12.6.1 Connection to a GSM network ... 618
12.6.2 Applications of the CP 1242-7 ... 619
12.6.3 Other properties of the CP ... 621
12.6.4 Accessories ... 622
12.6.5 Configuration examples for telecontrol ... 623

13 Teleservice communication (SMTP e-mail) ... 627

13.1 TM_Mail transfer e-mail instruction ... 627

14 Online and diagnostic tools ... 633

14.1 Status LEDs ... 633
14.2 Going online and connecting to a CPU ... 635
14.3 Assigning a name to a PROFINET IO device online ... 636
14.4 Setting the IP address and time of day ... 638
14.5 Resetting to factory settings ... 638
14.6 CPU operator panel for the online CPU ... 639
14.7 Monitoring the cycle time and memory usage ... 640
14.8 Displaying diagnostic events in the CPU ... 640
14.9 Comparing offline and online CPUs ... 641
14.10 Monitoring and modifying values in the CPU ... 642
14.10.1 Going online to monitor the values in the CPU ... 642
14.10.2 Displaying status in the program editor ... 643
14.10.3 Capturing the online values of a DB to reset the start values ... 644
14.10.4 Using a watch table to monitor and modify values in the CPU ... 645
14.10.4.1 Using a trigger when monitoring or modifying PLC tags ... 646
14.10.4.2 Enabling outputs in STOP mode ... 647
14.10.5 Forcing values in the CPU ... 647
14.10.5.1 Using the force table ... 647
14.10.5.2 Operation of the Force function ... 648
14.11 Downloading in RUN mode ... 650
14.11.1 Prerequisites for "Download in RUN mode" ... 651
14.11.2 Changing your program in RUN mode ... 651
14.11.3 Downloading selected blocks ... 652
14.11.4 Downloading a single selected block with a compile error in another block ... 653
14.11.5 System reaction if the download process fails ... 654
14.11.6 Downloading the program in RUN mode ... 654

A Technical specifications ... 657

A.1 General Technical Specifications ... 657
A.2 CPU 1211C ... 663
A.2.1 General specifications and features ... 663
A.2.2 Digital inputs and outputs ... 667
A.2.3 Analog inputs ... 668
A.2.3.1 Step response of the built-in analog inputs of the CPU ... 669
A.2.3.2 Sample time for the built-in analog ports of the CPU ... 669
A.2.3.3 Measurement ranges of the analog inputs for voltage ... 669

A.2.4 Wiring diagrams 670

A.3 CPU 1212C 671
A.3.1 General specifications and features 671
A.3.2 Digital inputs and outputs 675
A.3.3 Analog inputs 676
A.3.3.1 Step response of the built-in analog inputs of the CPU 677
A.3.3.2 Sample time for the built-in analog ports of the CPU 677
A.3.3.3 Measurement ranges of the analog inputs for voltage 678
A.3.4 Wiring diagrams 679

A.4 CPU 1214C 680
A.4.1 General specifications and features 680
A.4.2 Digital inputs and outputs 684
A.4.3 Analog inputs 685
A.4.3.1 Step response of the built-in analog inputs of the CPU 686
A.4.3.2 Sample time for the built-in analog ports of the CPU 686
A.4.3.3 Measurement ranges of the analog inputs for voltage 687
A.4.4 CPU 1214C Wiring Diagrams 687

A.5 Digital signal modules (SMs) 689
A.5.1 SM 1221 Digital Input Specifications 689
A.5.2 SM 1222 8-Point Digital Output Specifications 690
A.5.3 SM 1222 16-Point Digital Output Specifications 692
A.5.4 SM 1223 Digital Input/Output VDC Specifications 695
A.5.5 SM 1223 Digital Input/Output AC Specifications 698

A.6 Analog signal modules (SMs) 700
A.6.1 SM 1231 analog input module specifications 700
A.6.2 SM 1232 analog output module specifications 702
A.6.3 SM 1234 analog input/output module specifications 704
A.6.4 Step response of the analog inputs 707
A.6.5 Sample time and update times for the analog inputs 707
A.6.6 Measurement ranges of the analog inputs for voltage 708
A.6.7 Output (AQ) measurement ranges for voltage and current (SB and SM) 708

A.7 Thermocouple and RTD signal modules (SMs) 709
A.7.1 SM 1231 Thermocouple 709
A.7.1.1 Basic operation for a thermocouple 711
A.7.1.2 Selection tables for the SM 1231 thermocouple 712
A.7.2 SM 1231 RTD 714
A.7.2.1 Selection tables for the SM 1231 RTD 717

A.8 Digital signal boards (SBs) 720
A.8.1 SB 1221 200 kHz digital input specifications 720
A.8.2 SB 1222 200 kHz digital output specifications 722
A.8.3 SB 1223 200 kHz digital input / output specifications 724
A.8.4 SB 1223 2 X 24 VDC input / 2 X 24 VDC output specifications 726

A.9 Analog signal boards (SBs) 728
A.9.1 SB 1231 1 analog input specifications 728
A.9.2 SB 1232 1 analog output specifications 730
A.9.3 Measurement ranges for analog inputs and outputs 732
A.9.3.1 Step response of the analog inputs 732
A.9.3.2 Sample time and update times for the analog inputs 732
A.9.3.3 Measurement ranges of the analog inputs for voltage 733

A.9.3.4 Output (AQ) measurement ranges for voltage and current (SB and SM) ... 733
A.9.4 Thermocouple SBs ... 734
A.9.4.1 SB 1231 1 analog thermocouple input specifications ... 734
A.9.4.2 Basic operation for a thermocouple ... 736
A.9.5 RTD SBs ... 738
A.9.5.1 SB 1231 1 analog RTD input specifications ... 738
A.9.5.2 Selection tables for the SB 1231 RTD ... 740
A.10 Communication interfaces ... 743
A.10.1 PROFIBUS ... 743
A.10.1.1 CM 1242-5 ... 743
A.10.1.2 CM 1243-5 ... 744
A.10.2 GPRS ... 746
A.10.2.1 CP 1242-7 ... 747
A.10.3 CM 1243-2 AS-i Master ... 749
A.10.3.1 Technical data for the AS-i master CM 1243-2 ... 749
A.10.3.2 Electrical connections of the AS-i master CM 1243-2 ... 750
A.10.4 RS232, RS422, and RS485 ... 752
A.10.4.1 CB 1241 RS485 Specifications ... 752
A.10.4.2 CM 1241 RS485 Specifications ... 754
A.10.4.3 CM 1241 RS232 Specifications ... 755
A.10.4.4 CM 1241 RS422/485 Specifications ... 756
A.11 TeleService (TS Adapter and TS Adapter modular) ... 758
A.12 SIMATIC memory cards ... 758
A.13 Input simulators ... 758
A.14 I/O expansion cable ... 759
A.15 Companion products ... 760
A.15.1 PM 1207 power module ... 760
A.15.2 CSM 1277 compact switch module ... 760
B Calculating a power budget ... 761
C Order numbers ... 765
C.1 CPU modules ... 765
C.2 Signal modules (SMs) and signal boards (SBs) ... 765
C.3 Communication ... 766
C.4 Other modules ... 767
C.5 Memory cards ... 768
C.6 Basic HMI devices ... 768
C.7 Spare parts and other hardware ... 768
C.8 Programming software ... 769
C.9 Documentation ... 769
Index ... 771

Product overview

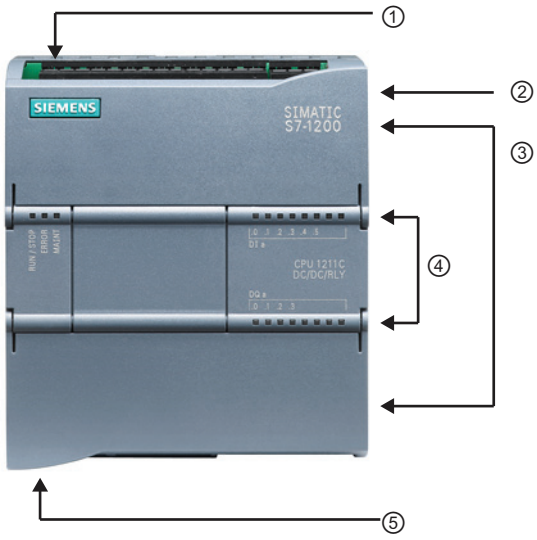
1

1.1 Introducing the S7-1200 PLC

The S7-1200 controller provides the flexibility and power to control a wide variety of devices in support of your automation needs. The compact design, flexible configuration, and powerful instruction set combine to make the S7-1200 a perfect solution for controlling a wide variety of applications.

The CPU combines a microprocessor, an integrated power supply, input and output circuits, built-in PROFINET, high-speed motion control I/O, and on-board analog inputs in a compact housing to create a powerful controller. After you download your program, the CPU contains the logic required to monitor and control the devices in your application. The CPU monitors the inputs and changes the outputs according to the logic of your user program, which can include Boolean logic, counting, timing, complex math operations, and communications with other intelligent devices.

The CPU provides a PROFINET port for communication over a PROFINET network. Additional modules are available for communicating over PROFIBUS, GPRS, RS485 or RS232 networks.



① Power connector
② Memory card slot under top door
③ Removable user wiring connectors (behind the doors)
④ Status LEDs for the on-board I/O
⑤ PROFINET connector (on the bottom of the CPU)

Several security features help protect access to both the CPU and the control program:

- Every CPU provides password protection (Page 153) that allows you to configure access to the CPU functions.
- You can use "know-how protection" (Page 154) to hide the code within a specific block.
- You can use copy protection (Page 156) to bind your program to a specific memory card or CPU.

Table 1- 1 Comparing the CPU models

Feature		CPU 1211C	CPU 1212C	CPU 1214C
Physical size (mm)		90 x 100 x 75	90 x 100 x 75	110 x 100 x 75
User memory	Work	25 Kbytes	25 Kbytes	50 Kbytes
	Load	1 Mbyte	1 Mbyte	2 Mbytes
	Retentive	2 Kbytes	2 Kbytes	2 Kbytes
Local on-board I/O	Digital	6 inputs/4 outputs	8 inputs/6 outputs	14 inputs/10 outputs
	Analog	2 inputs	2 inputs	2 inputs
Process image size	Inputs (I)	1024 bytes	1024 bytes	1024 bytes
	Outputs (Q)	1024 bytes	1024 bytes	1024 bytes
Bit memory (M)		4096 bytes	4096 bytes	8192 bytes
Signal module (SM) expansion		None	2	8
Signal board (SB) or communication board (CB)		1	1	1
Communication module (CM) (left-side expansion)		3	3	3
High-speed counters	Total	3	4	6
	Single phase	3 at 100 kHz	3 at 100 kHz 1 at 30 kHz	3 at 100 kHz 3 at 30 kHz
	Quadrature phase	3 at 80 kHz	3 at 80 kHz 1 at 20 kHz	3 at 80 kHz 3 at 20 kHz
Pulse outputs[1]		2	2	2
Memory card		SIMATIC Memory card (optional)		
Real time clock retention time		10 days, typical / 6 day minimum at 40 degrees C		
PROFINET		1 Ethernet communications port		
Real math execution speed		18 µs/instruction		
Boolean execution speed		0.1 µs/instruction		

[1] For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

Each CPU provides dedicated HMI connections to support up to 3 HMI devices. The total number of HMI is affected by the types of HMI panels in your configuration. For example, you could have up to three SIMATIC Basic panels connected to your CPU, or you could have up to two SIMATIC Comfort panels with one additional Basic panel.

The different CPU models provide a diversity of features and capabilities that help you create effective solutions for your varied applications. For detailed information about a specific CPU, see the technical specifications (Page 657).

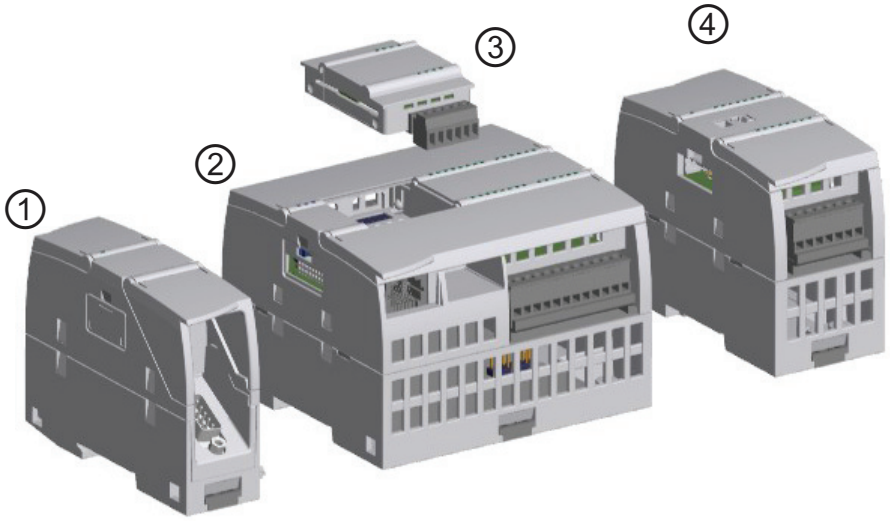
Table 1- 2 Blocks, timers and counters supported by S7-1200

Element		Description
Blocks	Type	OB, FB, FC, DB
	Size	25 Kbytes (CPU 1211C and CPU 1212C) 50 Kbytes (CPU 1214C)
	Quantity	Up to 1024 blocks total (OBs + FBs + FCs + DBs)
	Address range for FBs, FCs, and DBs	1 to 65535 (such as FB 1 to FB 65535)
	Nesting depth	16 from the program cycle or start up OB; 4 from the time delay interrupt, time-of-day interrupt, cyclic interrupt, hardware interrupt, time error interrupt, or diagnostic error interrupt OB
	Monitoring	Status of 2 code blocks can be monitored simultaneously
OBs	Program cycle	Multiple: OB 1, OB 200 to OB 65535
	Startup	Multiple: OB 100, OB 200 to OB 65535
	Time-delay interrupts and cyclic interrupts	4[1] (1 per event): OB 200 to OB 65535
	Hardware interrupts (edges and HSC)	50 (1 per event): OB 200 to OB 65535
	Time error interrupts	1: OB 80
	Diagnostic error interrupts	1: OB 82
Timers	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, 16 bytes per timer
Counters	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, size dependent upon count type • SInt, USInt: 3 bytes • Int, UInt: 6 bytes • DInt, UDInt: 12 bytes

[1] Time-delay and cyclic interrupts use the same resources in the CPU. You can have only a total of 4 of these interrupts (time-delay plus cyclic interrupts). You cannot have 4 time-delay interrupts and 4 cyclic interrupts.

1.2 Expansion capability of the CPU

The S7-1200 family provides a variety of modules and plug-in boards for expanding the capabilities of the CPU with additional I/O or other communication protocols. For detailed information about a specific module, see the technical specifications (Page 657).



① Communication module (CM), communcation processor (CP), or TS Adapter
② CPU
③ Signal board (SB) or communication board (CB)
④ Signal module (SM)

Table 1- 3 Digital signal modules and signal boards

Type	Input only	Output only	Combination In/Out
③ digital SB	• 4 x 24VDC In, 200 kHz • 4 x 5VDC In, 200 kHz	• 4 x 24VDC Out, 200 kHz • 4 x 5VDC Out, 200 kHz	• 2 x 24VDC In / 2 x 24VDC Out • 2 x 24VDC In / 2 x 24VDC Out, 200 kHz • 2 x 5VDC In / 2 x 5VDC Out, 200 kHz
④ digital SM	• 8 x 24VDC In	• 8 x 24VDC Out • 8 x Relay Out • 8 x Relay Out (Changeover)	• 8 x 24VDC In / 8 x 24VDC Out • 8 x 24VDC In / 8 x Relay Out • 8 x 120/230VAC In / 8 x Relay Out
	• 16 x 24VDC In	• 16 x 24VDC Out • 16 x Relay Out	• 16 x 24VDC In / 16 x 24VDC Out • 16 x 24VDC In / 16 x Relay Out

Table 1- 4 Analog signal modules and signal boards

Type	Input only	Output only	Combination In/Out
③ analog SB	• 1 x 12 bit Analog In • 1 x 16 bit RTD • 1 x 16 bit Thermocouple	• 1 x Analog Out	-
④ analog SM	• 4 x Analog In • 8 x Analog In • Thermocouple: – 4 x 16 bit TC – 8 x 16 bit TC • RTD: – 4 x 16 bit RTD – 8 x 16 bit RTD	• 2 x Analog Out • 4 x Analog Out	• 4 x Analog In / 2 x Analog Out

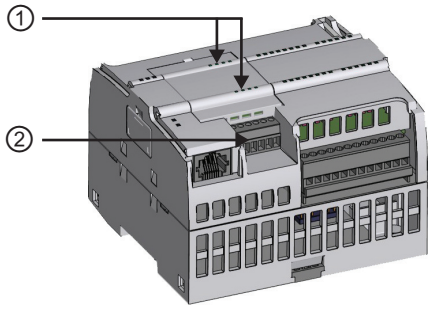
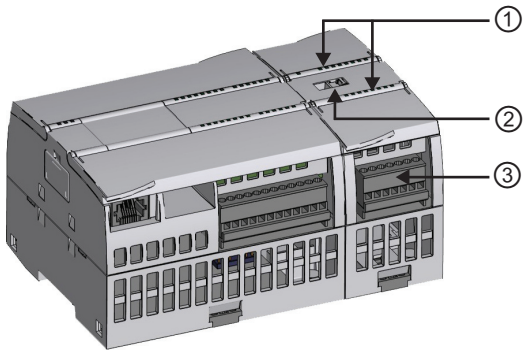
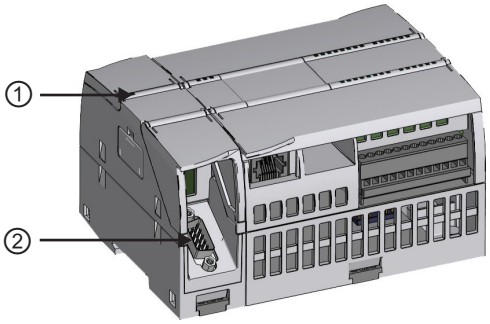
Table 1- 5 Communication interfaces

Module	Type	Description
① Communication module (CM)	RS232	Full-duplex
	RS485	Half-duplex
	RS422/485	Full-duplex (RS422) Half-duplex (RS485)
	PROFIBUS Master	DPV1
	PROFIBUS Slave	DPV1
	AS-i Master (CM 1243-2)	AS-Interface
① Communication processor (CP)	Modem connectivity	GPRS
① Communication board (CB)	RS485	Half-duplex
① TeleService	TS Adapter IE Basic[1]	Connection to CPU
	TS Adapter GSM	GSM/GPRS
	TS Adapter Modem	Modem
	TS Adapter ISDN	ISDN
	TS Adapter RS232	RS232

[1] The TS Adapter allows you to connect various communication interfaces to the PROFINET port of the CPU. You install the TS Adapter on the left side of the CPU and connect the TS Adapter modular (up to 3) onto the TS Adapter.

1.3 S7-1200 modules

Table 1- 6 S7-1200 expansion modules

Type of module	Description		
The CPU supports one plug-in expansion board: • A signal board (SB) provides additional I/O for your CPU. The SB connects on the front of the CPU. • A communication board (CB) allows you to add another communication port to your CPU.		① ②	Status LEDs on the SB Removable user wiring connector
Signal modules (SMs) add additional functionality to the CPU. SMs connect to the right side of the CPU. • Digital I/O • Analog I/O • RTD and thermocouple		① ② ③	Status LEDs Bus connector Removable user wiring connector
Communication modules (CMs) and communications processors (CPs) add communication options to the CPU, such as for PROFIBUS or RS232 / RS485 connectivity (for PtP, Modbus or USS), or the AS-i master. A CP provides capabilities for other types of communication, such as to connect the CPU over a GPRS network. • The CPU supports up to 3 CMs or CPs • Each CM or CP connects to the left side of the CPU (or to the left side of another CM or CP)		① ②	Status LEDs Communication connector

1.4 New features for S7-1200 and STEP 7 V11

STEP 7 V11 and the S7-1200 CPU firmware V2.2 provide additional capabilities and features.

- To allow you more control of how you define the data in your user program, S7-1200 provides additional data types, such as pointers, indexed arrays, and structures.
- STEP 7 V11 performs implicit data type conversions for instructions where smaller data types (such as SInt or Byte) are automatically converted to larger data types (such as DInt, DWord, Real or LReal). For example, an integer value (Int) will be converted automatically to a double integer (DInt) or a Real by an instruction that is configured to use a DInt or Real. You do not use an extra conversion instruction to convert the value.
- The instruction set has been expanded. New instructions include the following:
  - Communication instructions include S7 communication GET and PUT instructions, Distributed I/O RDREC, WRREC, and RALRM instructions, new PROFINET TUSEND and TURCV instructions, and Teleservice GPRS and TM_MAIL instructions.
  - A new Calculate instruction allows you to enter an equation directly into your LAD or FBD program.
  - A new MC_CommandTable instruction executes a series of individual motions for a motor control axis that you can combine into a movement sequence. Individual motions are configured in a technology object command table for pulse train output (TO_CommandTable_PTO).
  - Additional Interrupt instructions allow you to set and query time delay and cyclic interrupts.
  - You can also use the new diagnostic instructions to read the LED status or other diagnostic information about modules and devices.
  - There is also a new easy-to-use PID_3Step instruction.
- In your program logic, you can use a variable as an array index to access an individual array element (Page 197) in LAD, FBD, and SCL. Accessing an array through a variable is more direct than using the LAD/FBD FieldRead and FieldWrite instructions, which still exist to provide read and write access to the individual elements in an array.
- The GET/PUT instructions connection parameter assignment (Page 465) is a user aid that helps you configure S7-connections, using the property view page for the GET and PUT communication blocks that support CPU-CPU communication.
- The Download in RUN feature (Page 650) allows you to make program changes in RUN mode without stopping the CPU.
- PROFINET UDP is now supported. UDP provides a "broadcast" communications functionality.
- The S7-1200 CPU is a PROFINET IO controller.
- STEP 7 V11 provides an "undo" function.
- STEP 7 provides STOP and RUN buttons (Page 34) on the toolbar for stopping and starting the CPU.
- The "force table' (Page 647) is separate from the watch table and allows you to force inputs and outputs.

- You can copy-protect (Page 156) your user program or code blocks by binding them to a specific CPU or memory card.
- You can capture the values of a DB (Page 644) to set those values as the start values.
- With a click of a button, you can export the data from tables in STEP 7 (such as a PLC tag table or a watch table) into Microsoft Excel. You can also use CNTL-C and CNTL-V to copy and paste between STEP 7 and Microsoft Excel.
- Disconnecting I/O devices (Page 37) from the configured network without losing the configured device or having to reconfigure the network.
- Changing the assignment of a DB (Page 36) for an FB or an instruction (such as for changing the association of an FB from a single-instance DB to a multi-instance DB).
- The ability to access an individual bit, byte, or word ("slice") (Page 96) within a PLC tag, data block tag, or memory location of a larger size.
- The ability to overlay (Page 98) a data type parameter with a other types, an array, or structs.
- The S7-1200 automation system satisfies the requirements of the Korean Certification (KC Mark).

STEP 7 Basic and STEP 7 Professional programming packages

STEP 7 provides two programming packages to provide the features you need. Both include the text-based, high-level programming language SCL (Structured Control Language).

- STEP 7 Basic provides all of the tools required for your S7-1200 project.

  With the STEP 7 Basic package, you can connect your S7-1200 CPUs and the Basic HMI panels onto a PROFINET network. By adding a communication module (CM), communication processor (CP), or communication board (CB) to the device configuration of the CPU, you can connect to other types of networks, such as PROFIBUS or RS485.
- STEP 7 Professional expands S7-1200 to include the world of S7-300 and S7-400. You can now create networks using all of these SIMATIC controllers and I/O devices.

Web server functionality

To provide access to the CPU over the Internet, S7-1200 supports the S7 web server functionality, with standard web pages stored in the CPU memory. You can also create your own web pages for accessing data in the CPU.

Data logs

S7-1200 supports the creation of data log files to store process values. You use specific DataLog instructions for creating and managing the data logs. The data log files are stored in a standard CSV format, which can be opened with most spreadsheet applications.

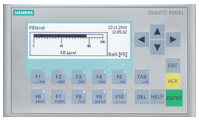
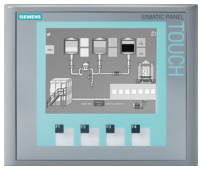
New modules for the S7-1200

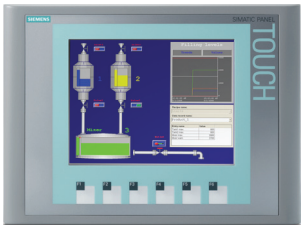
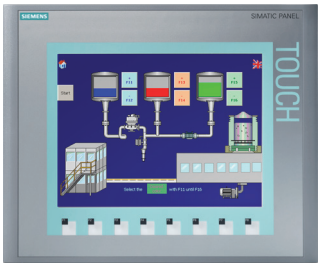
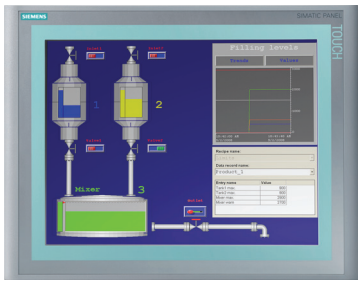
A variety of new modules expand the power of the S7-1200 CPU and to provide the flexibility to meet your automation needs:

- New I/O signal modules (SMs) including a new SM 1222 DQ8 RLY Changeover
- New signal boards (SBs) provide thermocouple (TC) and RTD capability
- New signal boards (SBs) provide high-speed (200 kHz) I/O.
- New communication modules (CMs) allow the S7-1200 to function as either a PROFIBUS master or slave device, and includes a new CM 1241 RS422/485.
- New communication interfaces support TeleService communication (modem, ISDN, GSM/GPRS, and RS232).
- A new communication board (CB) plugs into the front of the CPU to provide RS485 functionality.
- A new AS-i Master communication module, CM 1243-2 AS-i Master

1.5 Basic HMI panels

Because visualization is becoming a standard component for most machine designs, the SIMATIC HMI Basic Panels provide touch-screen devices for basic operator control and monitoring tasks. All panels have a protection rating for IP65 and have CE, UL, cULus, and NEMA 4x certification.

Basic HMI Panel	Description	Technical data
 KP 300 Basic PN	3.6" membrane keyboard with 10 freely configurable tactile keys • Mono (STN, black/white) • 87 mm x 31 mm (3.6") • Backlight color programmed (white, green, yellow or red) • Resolution: 240 x 80	• 250 tags • 50 process screens • 200 alarms • 25 curves • 40 KB recipe memory • 5 recipes, 20 data records, 20 entries
 KTP 400 Basic PN	4" touch screen with 4 tactile keys • Mono (STN, gray scale) • 76.79 mm x 57.59 mm (3.8") Portrait or landscape • Resolution: 320 x 240	• 250 tags • 50 process screens • 200 alarms • 25 curves • 40 KB recipe memory • 5 recipes, 20 data records, 20 entries

Basic HMI Panel	Description	Technical data
 KTP 600 Basic PN	6" touch screen with 6 tactile keys • Color (TFT, 256 colors) or Mono (STN, gray scales) • 115.2 mm x 86.4 mm (5.7") Portrait or landscape • Resolution: 320 x 240	• 500 tags • 50 process screens • 200 alarms • 25 curves • 40 KB recipe memory • 5 recipes, 20 data records, 20 entries
 KTP 1000 Basic PN	10" touch screen with 8 tactile keys • Color (TFT, 256 colors) • 211.2 mm x 158.4 mm (10.4") • Resolution: 640 x 480	• 500 tags • 50 process screens • 200 alarms • 25 curves • 40 KB recipe memory • 5 recipes, 20 data records, 20 entries
 TP 1500 Basic PN	15" touch screen • Color (TFT, 256 colors) • 304.1 mm x 228.1 mm (15.1") • Resolution: 1024 x 768	• 500 tags • 50 process screens • 200 alarms • 25 curves • 40 KB recipe memory (integrated flash) • 5 recipes, 20 data records, 20 entries

STEP 7 programming software

2

STEP 7 provides a user-friendly environment to develop, edit, and monitor the logic needed to control your application, including the tools for managing and configuring all of the devices in your project, such as controllers and HMI devices. To help you find the information you need, STEP 7 provides an extensive online help system.

STEP 7 provides standard programming languages for convenience and efficiency in developing the control program for your application.

- LAD (ladder logic) is a graphical programming language. The representation is based on circuit diagrams (Page 144).
- FBD (Function Block Diagram) is a programming language that is based on the graphical logic symbols used in Boolean algebra (Page 145).
- SCL (structured control language) is a text-based, high-level programming language.

When you create a code block, you select the programming language to be used by that block. Your user program can utilize code blocks created in any or all of the programming languages.

2.1 System requirements

To install the STEP 7 software on a PC running Windows XP, or Windows 7 operating system, you must log in with Administrator privileges.

Table 2- 1 System requirements

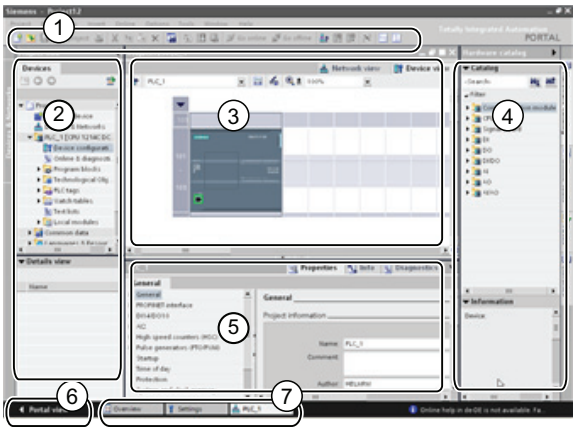
Hardware/software	Requirements
Processor type	Pentium M, 1.6 GHz or similar
RAM	1 GB
Available hard disk space	2 GB on system drive C:\
Operating systems	• Windows XP Professional SP3 • Windows 2003 Server R2 StdE SP2 • Windows 7 Home Premium (STEP 7 Basic only, not supported for STEP 7 Professional) • Windows 7 (Professional, Enterprise, Ultimate) • Windows 2008 Server StdE R2
Graphics card	32 MB RAM 24-bit color depth
Screen resolution	1024 x 768
Network	20 Mbit/s Ethernet or faster
Optical drive	DVD-ROM

2.2 Different views to make the work easier

STEP 7 provides a user-friendly environment to develop controller logic, configure HMI visualization, and setup network communication. To help increase your productivity, STEP 7 provides two different views of the project: a task-oriented set of portals that are organized on the functionality of the tools (Portal view), or a project-oriented view of the elements within the project (Project view). Choose which view helps you work most efficiently. With a single click, you can toggle between the Portal view and the Project view.

Portal view

① Portals for the different tasks
② Tasks for the selected portal
③ Selection panel for the selected action
④ Changes to the Project view



Project view

① Menus and toolbar
② Project navigator
③ Work area
④ Task cards
⑤ Inspector window
⑥ Changes to the Portal view
⑦ Editor bar

With all of these components in one place, you have easy access to every aspect of your project. For example, the inspector window shows the properties and information for the object that you have selected in the work area. As you select different objects, the inspector window displays the properties that you can configure. The inspector window includes tabs that allow you to see diagnostic information and other messages.

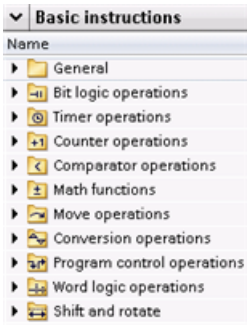
By showing all of the editors that are open, the editor bar helps you work more quickly and efficiently. To toggle between the open editors, simply click the different editor. You can also arrange two editors to appear together, arranged either vertically or horizontally. This feature allows you to drag and drop between editors.

2.3 Easy-to-use tools

2.3.1 Inserting instructions into your user program

STEP 7 provides task cards that contain the instructions for your program. The instructions are grouped according to function.

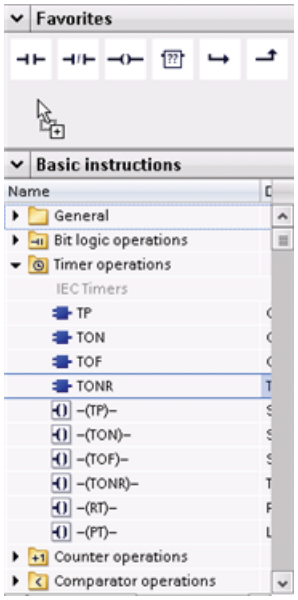
To create your program, you drag instructions from the task card onto a network.



2.3.2 Accessing instructions from the "Favorites" toolbar

STEP 7 provides a "Favorites" toolbar to give you quick access to the instructions that you frequently use. Simply click the icon for the instruction to insert it into your network!

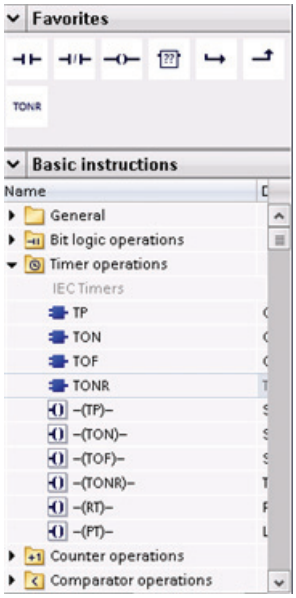
(For the "Favorites" in the instruction tree, double-click the icon.)



You can easily customize the "Favorites" by adding new instructions.

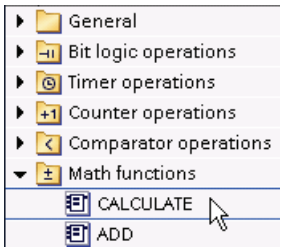
Simply drag and drop an instruction to the "Favorites".

The instruction is now just a click away!

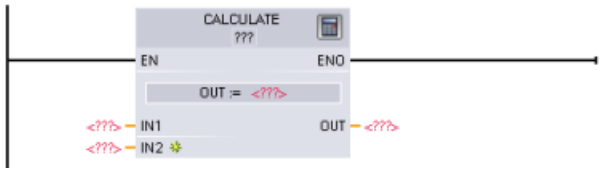


2.3.3 Creating a complex equation with a simple instruction

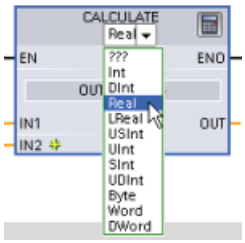
The Calculate instruction lets you create a math function that operates on multiple input parameters to produce the result, according to the equation that you define.



In the Basic instruction tree, expand the Math functions folder. Double-click the Calculate instruction to insert the instruction into your user program.



The unconfigured Calculate instruction provides two input parameters and an output parameter.



Click the "???" and select the data types for the input and output parameters. (The input and output parameters must all be the same data type.)

For this example, select the "Real" data type.

Click the "Edit equation" icon to enter the equation.

For this example, enter the following equation for scaling a raw analog value. (The "In" and "Out" designations correspond to the parameters of the Calculate instruction.)

$Out_{value} = ((Out_{high} - Out_{low}) / (In_{high} - In_{low})) * (In_{value} - In_{low}) + Out_{low}$

Out = ((in4 - in5) / (in2 - in3)) * (in1 - in3) + in5

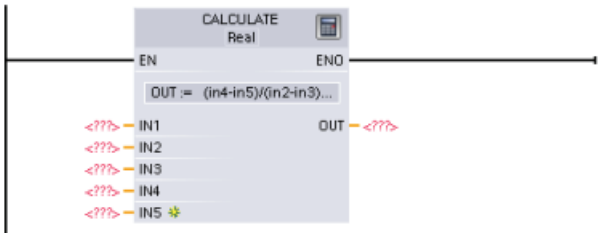
Where: Out_{value} (Out) Scaled output value

In value	(in1)	Analog input value
In high	(in2)	Upper limit for the scaled input value
In low	(in3)	Lower limit for the scaled input value
Out high	(in4)	Upper limit for the scaled output value
Out low	(in5)	Lower limit for the scaled ouput value

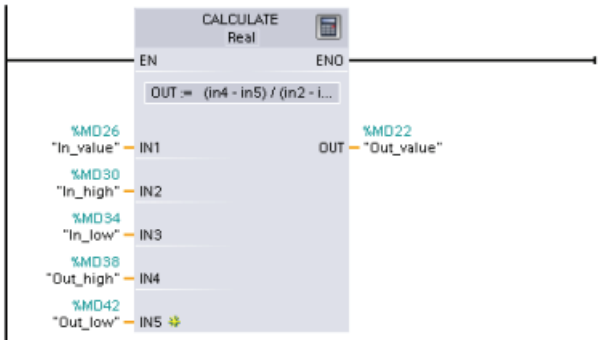
In the "Edit Calculate" box, enter the equation with the parameter names:

OUT = ((in4 - in5) / (in2 - in3)) * (in1 - in3) + in5

When you click "OK", the Calculate instruction creates the inputs required for the instruction.



Enter the tag names for the values that correspond to the parameters.



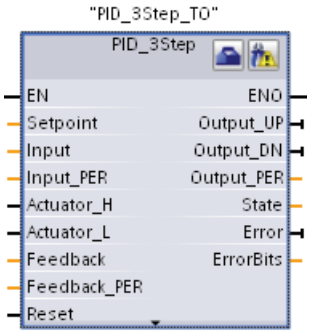
2.3.4 Adding inputs or outputs to a LAD or FBD instruction

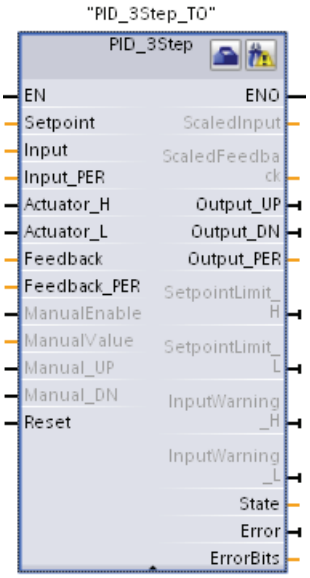
Some of the instructions allow you to create additional inputs or outputs.

- To add an input or output, click the "Create" icon or right-click on an input stub for one of the existing IN or OUT parameters and select the "Insert input" command.
- To remove an input or output, right-click on the stub for one of the existing IN or OUT parameters (when there are more than the original two inputs) and select the "Delete" command.

2.3.5 Expandable instructions

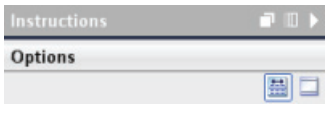
Some of the more complex instructions are expandable, displaying only the key inputs and outputs. To display the inputs and outputs, click the arrow at the bottom of the instruction.



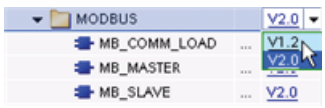


2.3.6 Selecting a version for an instruction

The development and release cycles for certain sets of instructions (such as Modbus, PID and motion) have created multiple released versions for these instructions. To help ensure compatibility and migration with older projects, STEP 7 allows you to choose which version of instruction to insert into your user program.

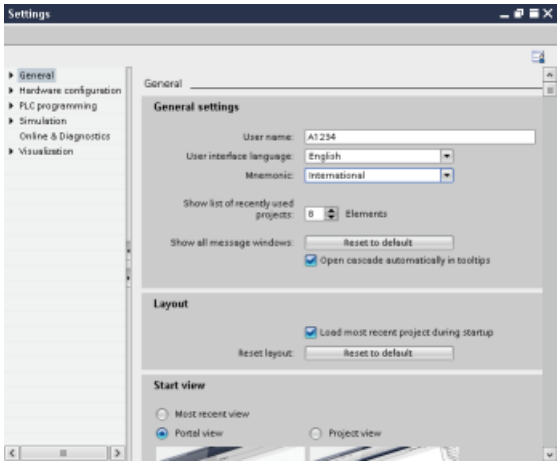


Click the icon on the instruction tree task card to enable the headers and columns of the instruction tree.



To change the version of the instruction, select the appropriate version from the drop-down list.

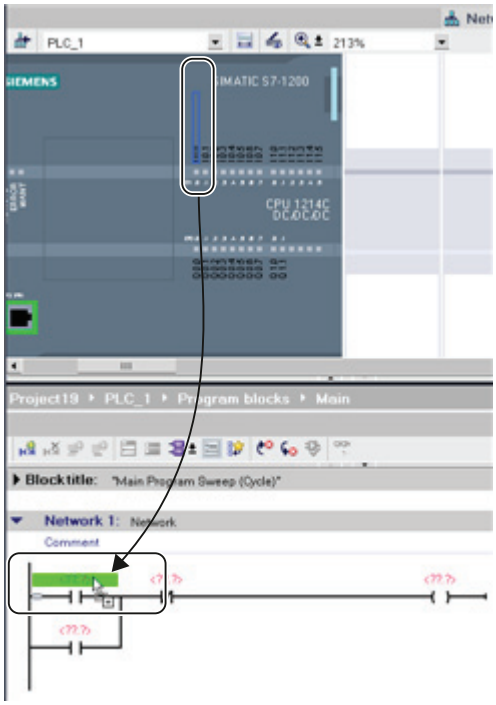
2.3.7 Modifying the appearance and configuration of STEP 7



You can select a variety of settings, such as the appearance of the interface, language, or the folder for saving your work.

Select the "Settings" command from the "Options" menu to change these settings.

2.3.8 Dragging and dropping between editors

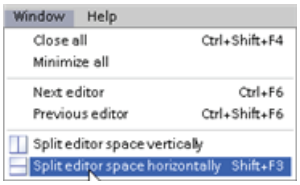


To help you perform tasks quickly and easily, STEP 7 allows you to drag and drop elements from one editor to another. For example, you can drag an input from the CPU to the address of an instruction in your user program.

You must zoom in at least 200% to select the inputs or outputs of the CPU.

Notice that the tag names are displayed not only in the PLC tag table, but also are displayed on the CPU.

To display two editors at one time, use the "Split editor" menu commands or buttons in the toolbar.



To toggle between the editors that have been opened, click the icons in the editor bar.

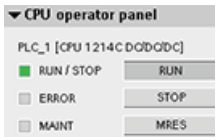
2.3.9 Changing the operating mode of the CPU

The CPU does not have a physical switch for changing the operating mode (STOP or RUN).

Use the "Start CPU" and "Stop CPU" toolbar buttons to change the operating mode of the CPU.

When you configure the CPU in the device configuration, you configure the start-up behavior in the properties of the CPU.

The "Online and diagnostics" portal also provides an operator panel for changing the operating mode of the online CPU. To use the CPU operator panel, you must be connected online to the CPU. The "Online tools" task card displays an operator panel that shows the operating mode of the online CPU. The operator panel also allows you to change the operating mode of the online CPU.



Use the button on the operator panel to change the operating mode (STOP or RUN). The operator panel also provides an MRES button for resetting the memory.

The color of the RUN/STOP indicator shows the current operating mode of the CPU. Yellow indicates STOP mode, and green indicates RUN mode.

2.3.10 Capturing and restoring a block state

STEP 7 provides a means for capturing the state of a code block to create a benchmark or reference point for the user program. A block state represents the status of a code block at a specific time. Generating a block state allows you to reset the block to this state at any time, discarding all changes you have since made. You can restore the user program to the state of the block, even if you have made and saved changes to the program.

You can capture up to 10 block states in your project. Block states are still accessible after the project has been saved. However, closing the project removes any captured block states.

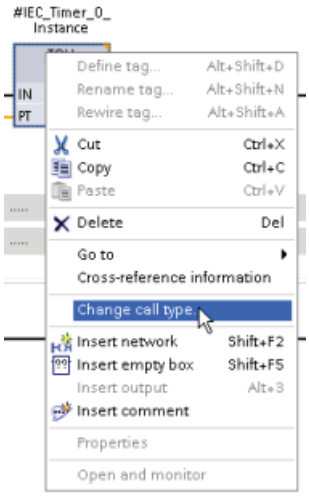
The ability to capture and restore the state of the program block is more powerful than the "Undo" function because the block state transcends the "Save" function.

Click the "Capture block state" button to save the current state of the user program. After you capture a state of the user program, the program block displays a "Block state" icon.

Click the "Restore block state" button to restore the program block to the block state that was captured.

Click the "Delete block state" button to remove the block state that was captured.

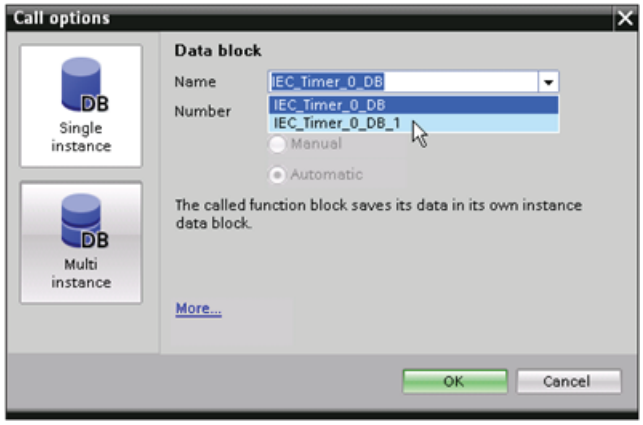
2.3.11 Changing the call type for a DB



STEP 7 allows you to easily create or change the association of a DB for an instruction or an FB that is in an FB.

- You can switch the association between different DBs.
- You can switch the association between a single-instance DB and a multi-instance DB.
- You can create an instance DB (if an instance DB is missing or not available).

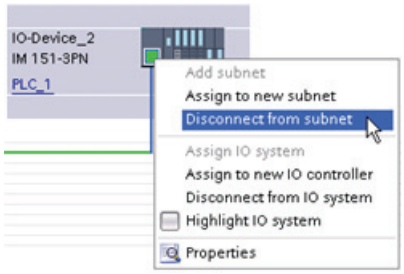
You can access the "Change call type" command either by right-clicking the instruction or FB in the program editor or by selecting the "Block call" command from the "Options" menu.



The "Call options" dialog allows you to select a single-instance or multi-instance DB. You can also select specific DBs from a drop-down list of available DBs.

2.3.12 Temporarily disconnecting devices from a network

You can disconnect individual network devices from the subnet. Because the configuration of the device is not removed from the project, you can easily restore the connection to the device.



Right-click the interface port of the network device and select the "Disconnect from subnet" command from the context menu.

STEP 7 reconfigures the network connections, but does not remove the disconnected device from the project. While the network connection is deleted, the interface addresses are not changed.

When you download the new network connections, the CPU goes to STOP mode.

To reconnect the device, simply create a new network connection to the port of the device.

2.3.13 Virtual unplugging of devices from the configuration

STEP 7 provides a storage area for "unplugged" modules. You can drag a module from the rack to save the configuration of that module. These unplugged modules are saved with your project, allowing you to reinsert the module in the future without having to reconfigure the parameters.

One use of this feature is for temporary maintenance. Consider a scenario where you might be waiting for a replacement module and plan to temporarily use a different module as a short-term replacement. You could drag the configured module from the rack to the "Unplugged modules" and then insert the temporary module.

Installation 3

3.1 Guidelines for installing S7-1200 devices

The S7-1200 equipment is designed to be easy to install. You can install an S7-1200 either on a panel or on a standard rail, and you can orient the S7-1200 either horizontally or vertically. The small size of the S7-1200 allows you to make efficient use of space.

⚠ WARNING

The SIMATIC S7-1200 PLCs are Open Type Controllers. It is required that you install the S7-1200 in a housing, cabinet, or electric control room. Entry to the housing, cabinet, or electric control room should be limited to authorized personnel.

Failure to follow these installation requirements could result in death, severe personal injury and/or property damage.

Always follow these requirements when installing S7-1200 PLCs.

Separate the S7-1200 devices from heat, high voltage, and electrical noise

As a general rule for laying out the devices of your system, always separate the devices that generate high voltage and high electrical noise from the low-voltage, logic-type devices such as the S7-1200.

When configuring the layout of the S7-1200 inside your panel, consider the heat-generating devices and locate the electronic-type devices in the cooler areas of your cabinet. Reducing the exposure to a high-temperature environment will extend the operating life of any electronic device.

Consider also the routing of the wiring for the devices in the panel. Avoid placing low-voltage signal wires and communications cables in the same tray with AC power wiring and high-energy, rapidly-switched DC wiring.

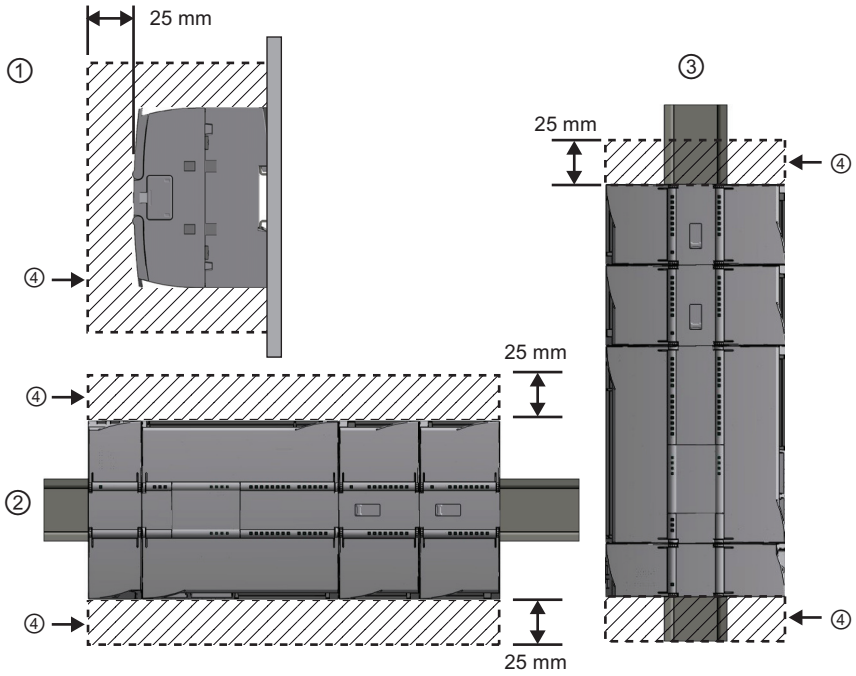
Provide adequate clearance for cooling and wiring

S7-1200 devices are designed for natural convection cooling. For proper cooling, you must provide a clearance of at least 25 mm above and below the devices. Also, allow at least 25 mm of depth between the front of the modules and the inside of the enclosure.

⚠ CAUTION

For vertical mounting, the maximum allowable ambient temperature is reduced by 10 degrees C. Orient a vertically mounted S7-1200 system as shown in the following figure.

When planning your layout for the S7-1200 system, allow enough clearance for the wiring and communications cable connections.



① Side view
② Horizontal installation
③ Vertical installation
④ Clearance area

3.2 Power budget

Your CPU has an internal power supply that provides power for the CPU, the signal modules, signal board and communication modules and for other 24 VDC user power requirements.

Refer to the technical specifications (Page 657) for information about the 5 VDC logic budget supplied by your CPU and the 5 VDC power requirements of the signal modules, signal boards, and communication modules. Refer to the "Calculating a power budget" (Page 761) to determine how much power (or current) the CPU can provide for your configuration.

The CPU provides a 24 VDC sensor supply that can supply 24 VDC for input points, for relay coil power on the signal modules, or for other requirements. If your 24 VDC power requirements exceed the budget of the sensor supply, then you must add an external 24 VDC power supply to your system. Refer to the technical specifications (Page 657) for the 24 VDC sensor supply power budget for your particular CPU.

Note

The CM 1243-5 (PROFIBUS master module) requires power from the 24 VDC sensor supply of the CPU.

If you require an external 24 VDC power supply, ensure that the power supply is not connected in parallel with the sensor supply of the CPU. For improved electrical noise protection, it is recommended that the commons (M) of the different power supplies be connected.

⚠ WARNING

Connecting an external 24 VDC power supply in parallel with the 24 VDC sensor supply can result in a conflict between the two supplies as each seeks to establish its own preferred output voltage level.

The result of this conflict can be shortened lifetime or immediate failure of one or both power supplies, with consequent unpredictable operation of the PLC system. Unpredictable operation could result in death, severe personal injury and/or property damage.

The DC sensor supply and any external power supply should provide power to different points.

Some of the 24 VDC power input ports in the S7-1200 system are interconnected, with a common logic circuit connecting multiple M terminals. For example, the following circuits are interconnected when designated as "not isolated" in the data sheets: the 24 VDC power supply of the CPU, the power input for the relay coil of an SM, or the power supply for a non-isolated analog input. All non-isolated M terminals must connect to the same external reference potential.

⚠ WARNING

Connecting non-isolated M terminals to different reference potentials will cause unintended current flows that may cause damage or unpredictable operation in the PLC and any connected equipment.

Failure to comply with these guidelines could cause damage or unpredictable operation which could result in death or serve personal injury and/or property damage.

Always ensure that all non-isolated M terminals in an S7-1200 system are connected to the same reference potential.

3.3 Installation and removal procedures

3.3.1 Mounting dimensions for the S7-1200 devices

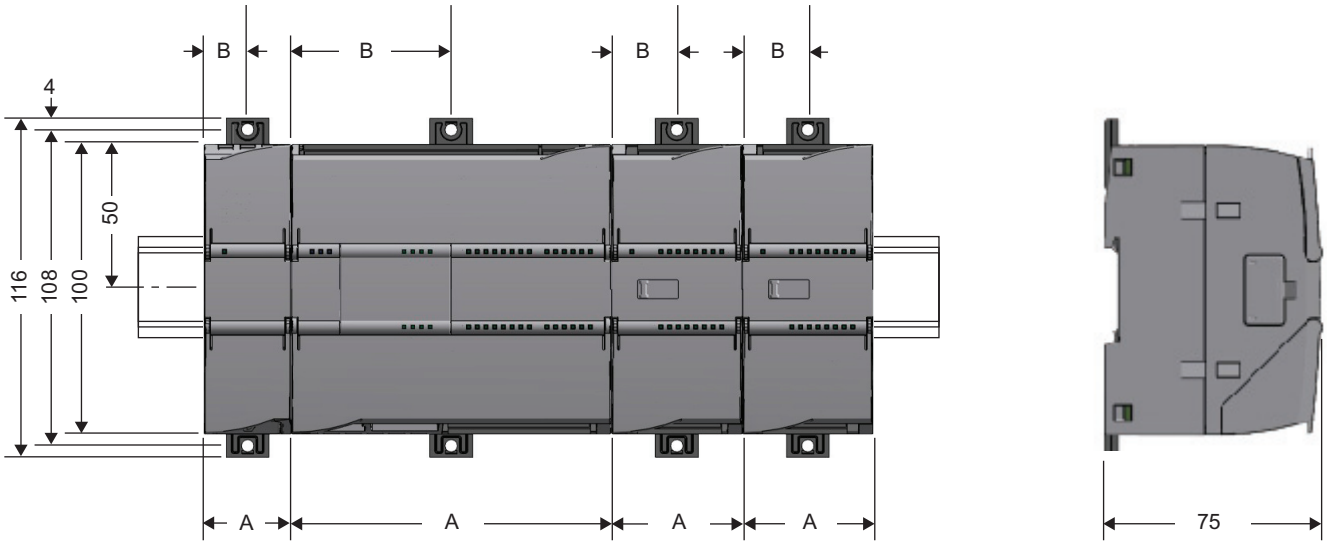


Table 3- 1 Mounting dimensions (mm)

S7-1200 Devices		Width A	Width B
CPU	CPU 1211C and CPU 1212C	90 mm	45 mm
	CPU 1214C	110 mm	55 mm
Signal modules	Digital 8 and 16 point Analog 2, 4, and 8 point Thermocouple 4 and 8 point RTD 4 point	45 mm	22.5 mm
	Digital DQ 8 x Relay (Changeover)	70	22.5
	Analog 16 point RTD 8 point	70 mm	35 mm
Communication interfaces	CM 1241 RS232, CM 1241 RS485 and CM 1241 RS422/485 CM 1243-5 PROFIBUS master and CM 1242-5 PROFIBUS slave CM 1242-2 AS-i Master CP 1242-7 GPRS	30 mm	15 mm
	TS AdapterIE Basic	60 mm [1]	15 mm

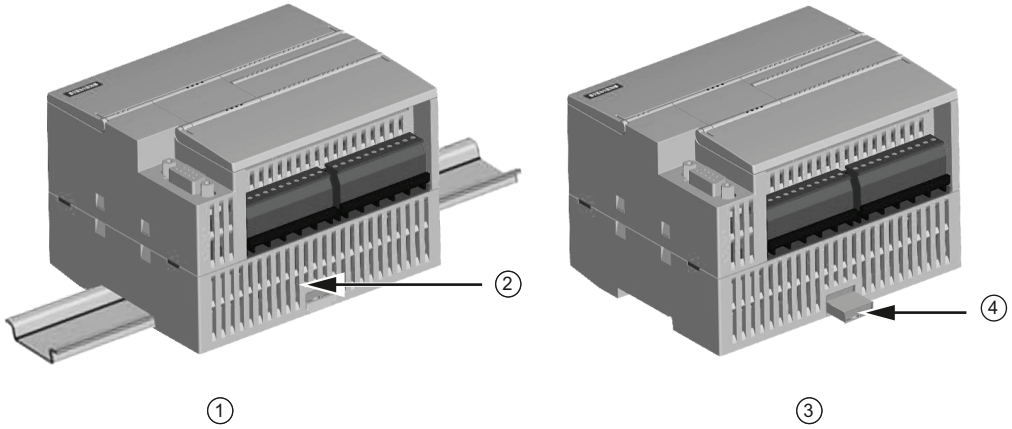
[1] Because you must install a TS Adapter modular with the TS Adapter, the total width ("width A") is 60 mm.

Each CPU, SM, CM, and CP supports mounting on either a DIN rail or on a panel. Use the DIN rail clips on the module to secure the device on the rail. These clips also snap into an extended position to provide screw mounting positions to mount the unit directly on a panel. The interior dimension of the hole for the DIN clips on the device is 4.3 mm.

A 25 mm thermal zone must be provided above and below the unit for free air circulation.

Installing and removing the S7-1200 devices

The CPU can be easily installed on a standard DIN rail or on a panel. DIN rail clips are provided to secure the device on the DIN rail. The clips also snap into an extended position to provide a screw mounting position for panel-mounting the unit.



① DIN rail installation
② DIN rail clip in latched position
③ Panel installation
④ Clip in extended position for panel mounting

Before you install or remove any electrical device, ensure that the power to that equipment has been turned off. Also, ensure that the power to any related equipment has been turned off.

⚠ WARNING

Installation or removal of S7-1200 or related equipment with the power applied could cause electric shock or unexpected operation of equipment.

Failure to disable all power to the S7-1200 and related equipment during installation or removal procedures could result in death, severe personal injury and/or property damage due to electric shock or unexpected equipment operation.

Always follow appropriate safety precautions and ensure that power to the S7-1200 is disabled before attempting to install or remove S7-1200 CPUs or related equipment.

Always ensure that whenever you replace or install an S7-1200 device you use the correct module or equivalent device.

WARNING

Incorrect installation of an S7-1200 module may cause the program in the S7-1200 to function unpredictably.

Failure to replace an S7-1200 device with the same model, orientation, or order could result in death, severe personal injury and/or property damage due to unexpected equipment operation.

Replace an S7-1200 device with the same model, and be sure to orient and position it correctly.

3.3.2 Installing and removing the CPU

You can install the CPU on a panel or on a DIN rail.

Note

Attach any communication modules to the CPU and install the assembly as a unit. Install signal modules separately after the CPU has been installed.

Consider the following when installing the units on the DIN rail or on a panel:

- For DIN rail mounting, make sure the upper DIN rail clip is in the latched (inner) position and that the lower DIN rail clip is in the extended position for the CPU and attached CMs.
- After installing the devices on the DIN rail, move the lower DIN rail clips to the latched position to lock the devices on the DIN rail.
- For panel mounting, make sure the DIN rail clips are pushed to the extended position.

To install the CPU on a panel, follow these steps:

1. Locate, drill, and tap the mounting holes (M4 or American Standard number 8), using the dimensions shown in the mounting dimensions.
2. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power.
3. Extend the mounting clips from the module. Make sure the DIN rail clips on the top and bottom of the CPU are in the extended position.
4. Secure the module to the panel, using screws placed into the clips.

Note

If your system is subject to a high vibration environment, or is mounted vertically, panel mounting the S7-1200 will provide a greater level of protection.

Table 3- 2 Installing the CPU on a DIN rail

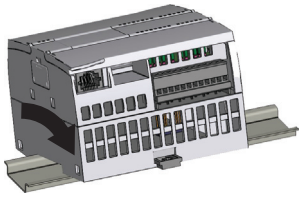
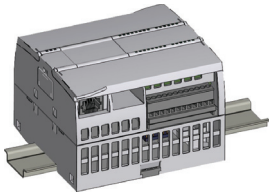
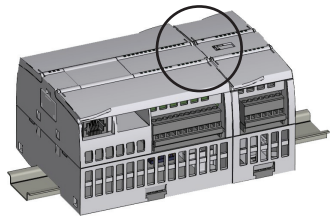
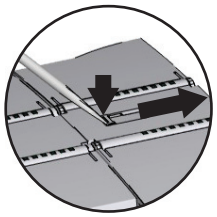
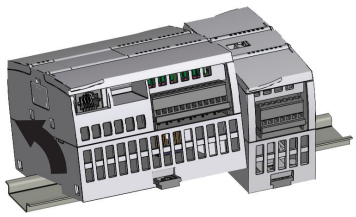
Task	Procedure
 	1. Install the DIN rail. Secure the rail to the mounting panel every 75 mm. 2. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 3. Hook the CPU over the top of the DIN rail. 4. Pull out the DIN rail clip on the bottom of the CPU to allow the CPU to fit over the rail. 5. Rotate the CPU down into position on the rail. 6. Push in the clips to latch the CPU to the rail.

Table 3- 3 Removing the CPU from a DIN rail

Task	Procedure	
		1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Disconnect the I/O connectors, wiring, and cables from the CPU (Page 50). 3. Remove the CPU and any attached communication modules as a unit. All signal modules should remain installed. 4. If an SM is connected to the CPU, retract the bus connector: – Place a screwdriver beside the tab on the top of the signal module. – Press down to disengage the connector from the CPU. – Slide the tab fully to the right. 5. Remove the CPU: – Pull out the DIN rail clip to release the CPU from the rail. – Rotate the CPU up and off the rail, and remove the CPU from the system.
		

3.3.3 Installing and removing an SB or a CB

Table 3- 4 Installing an SB or a CB

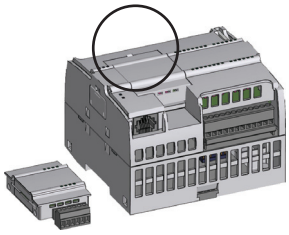
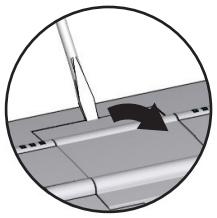
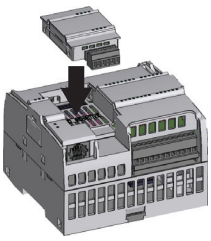
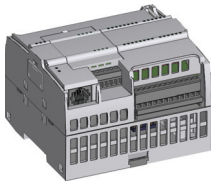
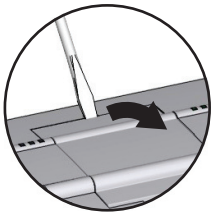
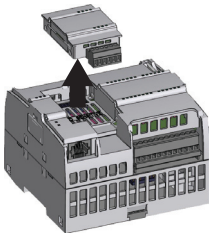
Task	Procedure
  	1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Remove the top and bottom terminal block covers from the CPU. 3. Place a screwdriver into the slot on top of the CPU at the rear of the cover. 4. Gently pry the cover up and remove it from the CPU. 5. Place the module straight down into its mounting position in the top of the CPU. 6. Firmly press the module into position until it snaps into place. 7. Replace the terminal block covers.

Table 3- 5 Removing an SB or a CB

Task	Procedure
  	1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Remove the top and bottom terminal block covers from the CPU. 3. Place a screwdriver into the slot on top of the module. 4. Gently pry the module up to disengage it from the CPU. 5. Remove the module straight up from its mounting position in the top of the CPU. 6. Replace the cover onto the CPU. 7. Replace the terminal block covers.

3.3.4 Installing and removing an SM

Table 3- 6 Installing an SM

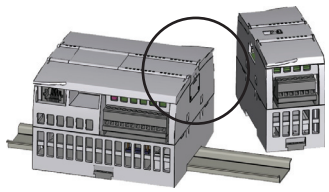
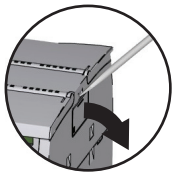
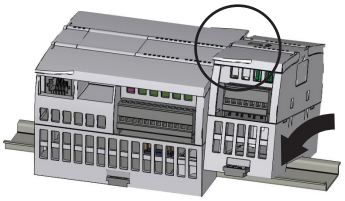
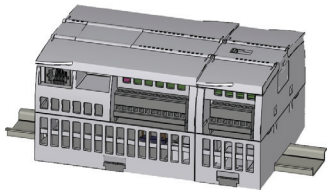
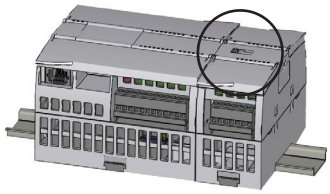
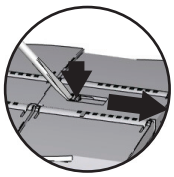
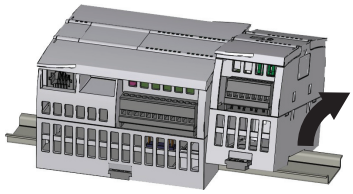
Task	Procedure	
		Install your SM after installing the CPU. 1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Remove the cover for the connector from the right side of the CPU. 3. Insert a screwdriver into the slot above the cover. 4. Gently pry the cover out at its top and remove the cover. Retain the cover for reuse.
		Connect the SM to the CPU: 1. Position the SM beside the CPU. 2. Hook the SM over the top of the DIN rail. 3. Pull out the bottom DIN rail clip to allow the SM to fit over the rail. 4. Rotate the SM down into position beside the CPU and push the bottom clip in to latch the SM onto the rail.
	Extending the bus connector makes both mechanical and electrical connections for the SM. 1. Place a screwdriver beside the tab on the top of the SM. 2. Slide the tab fully to the left to extend the bus connector into the CPU. Follow the same procedure to install a signal module to a signal module.	

Table 3- 7 Removing an SM

Task	Procedure
	You can remove any SM without removing the CPU or other SMs in place. 1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Remove the I/O connectors and wiring from the SM (Page 50). 3. Retract the bus connector. – Place a screwdriver beside the tab on the top of the SM. – Press down to disengage the connector from the CPU. – Slide the tab fully to the right. If there is another SM to the right, repeat this procedure for that SM.
	
	Remove the SM: 1. Pull out the bottom DIN rail clip to release the SM from the rail. 2. Rotate the SM up and off the rail. Remove the SM from the system. 3. If required, cover the bus connector on the CPU to avoid contamination. Follow the same procedure to remove a signal module from a signal module.

3.3.5 Installing and removing a CM or CP

Attach any communication modules to the CPU and install the assembly as a unit, as shown in Installing and removing the CPU (Page 44).

Table 3- 8 Installing a CM or CP

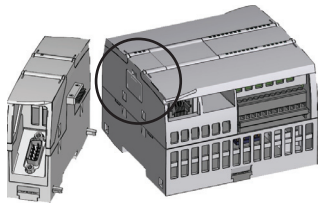
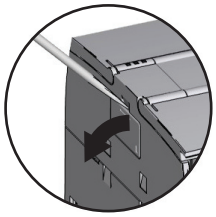
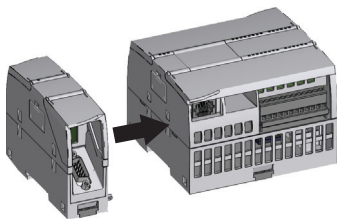
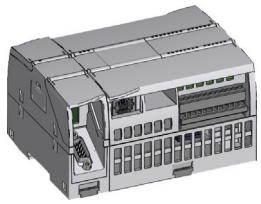
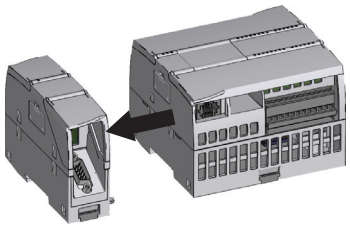
Task		Procedure
		1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Attach the CM to the CPU before installing the assembly as a unit to the DIN rail or panel. 3. Remove the bus cover from the left side of the CPU: – Insert a screwdriver into the slot above the bus cover. – Gently pry out the cover at its top. 4. Remove the bus cover. Retain the cover for reuse. 5. Connect the CM or CP to the CPU: – Align the bus connector and the posts of the CM with the holes of the CPU – Firmly press the units together until the posts snap into place. 6. Install the CPU and CP on a DIN rail or panel.
		

Table 3- 9 Removing a CM or CP

Task	Procedure
 	Remove the CPU and CM as a unit from the DIN rail or panel. 1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Remove the I/O connectors and all wiring and cables from the CPU and CMs. 3. For DIN rail mounting, move the lower DIN rail clips on the CPU and CMs to the extended position. 4. Remove the CPU and CMs from the DIN rail or panel. 5. Grasp the CPU and CMs firmly and pull apart.

CAUTION
Do not use a tool to separate the modules because this will damage the units.

3.3.6 Removing and reinstalling the S7-1200 terminal block connector

The CPU, SB and SM modules provide removable connectors to make connecting the wiring easy.

Table 3- 10 Removing the connector

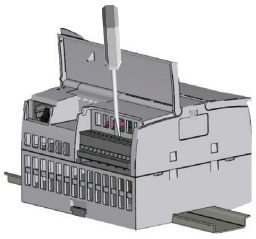
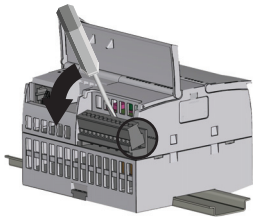
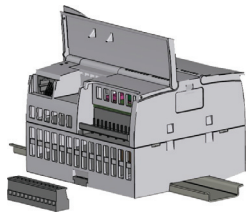
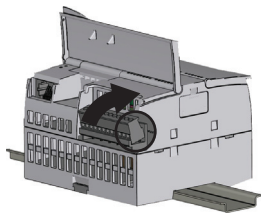
Task	Procedure
 	Prepare the system for terminal block connector removal by removing the power from the CPU and opening the cover above the connector. 1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Inspect the top of the connector and locate the slot for the tip of the screwdriver. 3. Insert a screwdriver into the slot. 4. Gently pry the top of the connector away from the CPU. The connector will release with a snap. 5. Grasp the connector and remove it from the CPU.

Table 3- 11 Installing the connector

Task	Procedure
 	Prepare the components for terminal block installation by removing power from the CPU and opening the cover for connector. 1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Align the connector with the pins on the unit. 3. Align the wiring edge of the connector inside the rim of the connector base. 4. Press firmly down and rotate the connector until it snaps into place. Check carefully to ensure that the connector is properly aligned and fully engaged.

3.3.7 Installing and removing the expansion cable

The S7-1200 expansion cable provides additional flexibility in configuring the layout of your S7-1200 system. Only one expansion cable is allowed per CPU system. You install the expansion cable either between the CPU and the first SM, or between any two SMs.

Table 3- 12 Installing and removing the male connector of the expansion cable

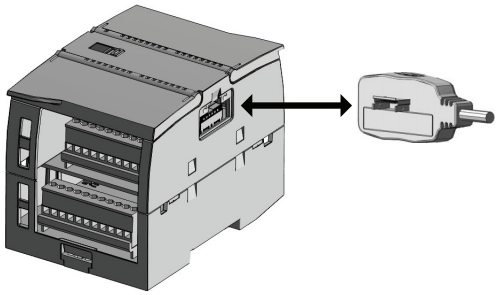
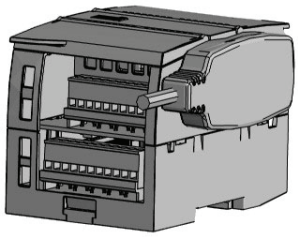
Task	Procedure
 	To install the male connector: 1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Push the connector into the bus connector on the right side of the signal module or CPU. To remove the male connector: 1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Pull out the male connector to release it from the signal module or CPU.

Table 3- 13 Installing the female connector of the expansion cable

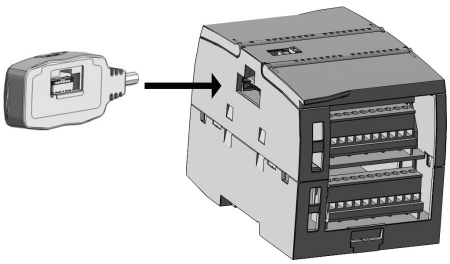
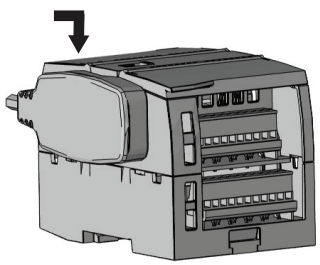
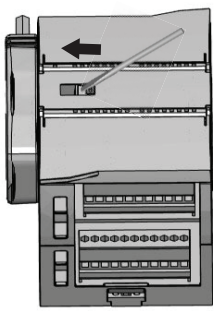
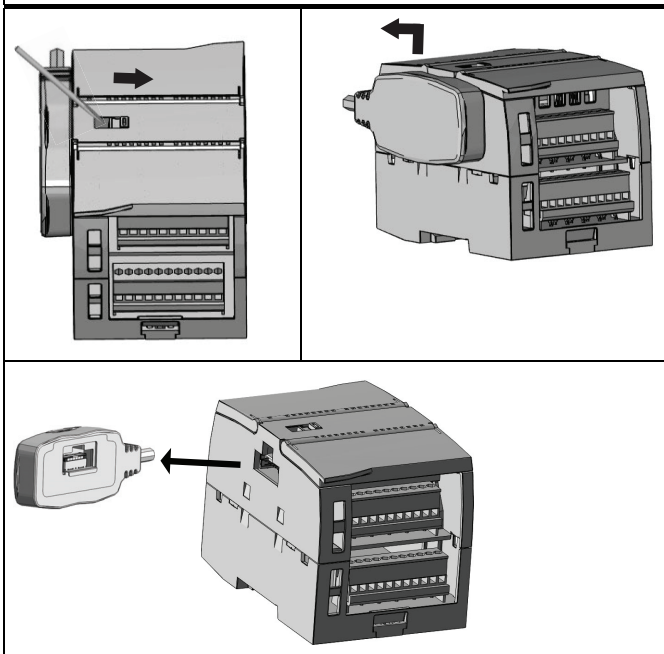
Task	Procedure
  	1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Place the female connector to the bus connector on the left side of the signal module. 3. Slip the hook extension of the female connector into the housing at the bus connector and press down slightly to engage the hook. 4. Lock the connector into place: – Place a screwdriver beside the tab on the top of the signal module. – Slide the tab fully to the left. To engage the connector, you must slide the connector tab all the way to the left. The connector tab must be locked into place.

Table 3- 14 Removing the female connector of the expansion cable

Task	Procedure
	1. Ensure that the CPU and all S7-1200 equipment are disconnected from electrical power. 2. Unlock the connector: – Place a screwdriver beside the tab on the top of the signal module. – Press down slightly and slide the tab fully to the right. 3. Lift the connector up slightly to disengage the hook extension. 4. Remove the female connector.

3.3.8 TS (teleservice) adapter

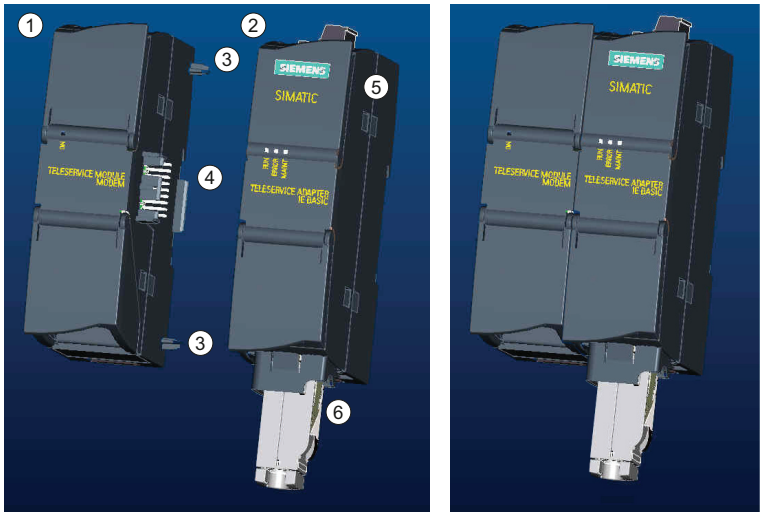
3.3.8.1 Connecting the TeleService Adapter

Before installing the TS (Teleservice) Adapter IE Basic, you must first connect the TS Adapter and a TS module.

Available TS modules:

- TS module RS232
- TS module Modem
- TS module GSM
- TS module ISDN

CAUTION
The TS module can be damaged if you touch the contacts of the plug connector ④of the TS module. Follow ESD guidelines in order to avoid damaging the TS module through electrostatic discharge. Before connecting a TS module and TS Adapter, make sure that both are in an idle state.



① TS module
② TS Adapter
③ Elements
④ Plug connector from the TS module
⑤ Cannot be opened
⑥ Ethernet port

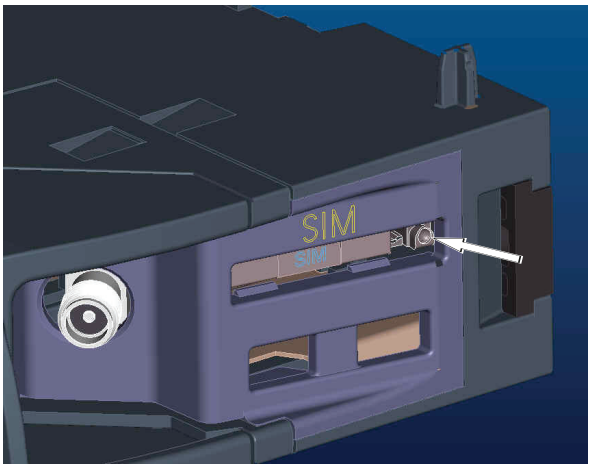
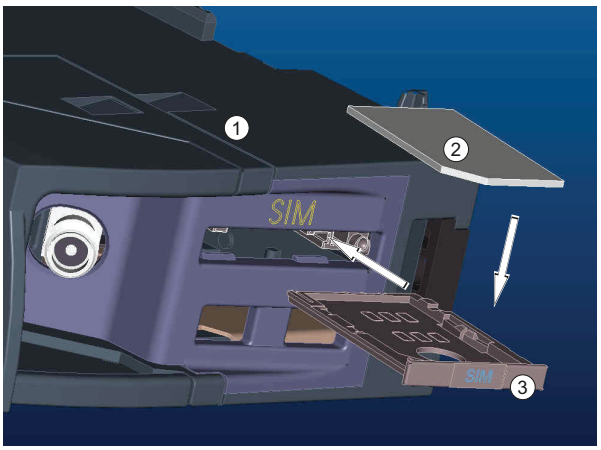
CAUTION
Before connecting a TS module and TS adapter basic unit, ensure that the contact pins ④ are not bent. When connecting, ensure that the male connector and guide elements are positioned correctly. Only connect a TS module into the TS adapter. Do not force a connection of the TS adapter to a different device, such as an S7-1200 CPU. Do not change the mechanical construction of the connector, and do not remove or damage the guide elements.

3.3.8.2 Installing the SIM card

Locate the SIM card slot on the underside of the TS module GSM.

NOTICE
The SIM card may only be removed or inserted if the TS module GSM is de-energized.

Table 3- 15 Installing the SIM card

Procedure	Task		
Use a sharp object to press the eject button of the SIM card tray (in the direction of the arrow) and remove the SIM card tray.			
Place the SIM card in the SIM card tray as shown and put the SIM card tray back into its slot.		①	TS Module GSM
		②	SIM card
		③	SIM card tray

Note

Ensure that the SIM card tray is correctly oriented in the card tray. Otherwise, the SIM card will not make connection with the module, and the eject button may not remove the card tray.

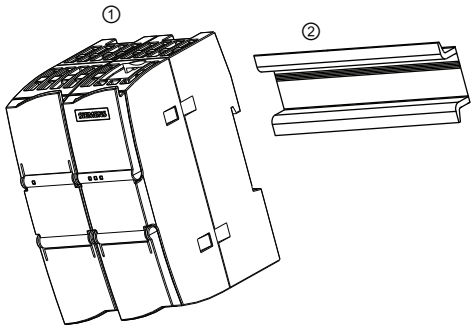
3.3.8.3 Installing the TS adapter unit

Prerequisites: You must have connected the TS Adapter and a TS module together, and the DIN rail must have been installed.

Note

If you install the TS unit vertically or in high-vibration environment, the TS module can become disconnected from the TS Adapter. Use an end bracket 8WA1 808 on the DIN rail to ensure that the modules remain connected.

Table 3- 16 Installing and removing the TS Adapter

Task	Procedure
	Installation: 1. Hook the TS Adapter with attached TS module ① on the DIN rail ②. 2. Rotate the unit back until it engages. 3. Push in the DIN rail clip on each module to attach the each module to the rail.
	Removal: 1. Remove the analog cable and Ethernet cable from the underside of the TS Adapter. 2. Remove power from the TS Adapter. 3. Use a screwdriver to disengage the rail clips on both modules. 4. Rotate the unit upwards to remove the unit from the DIN rail.

⚠ WARNING

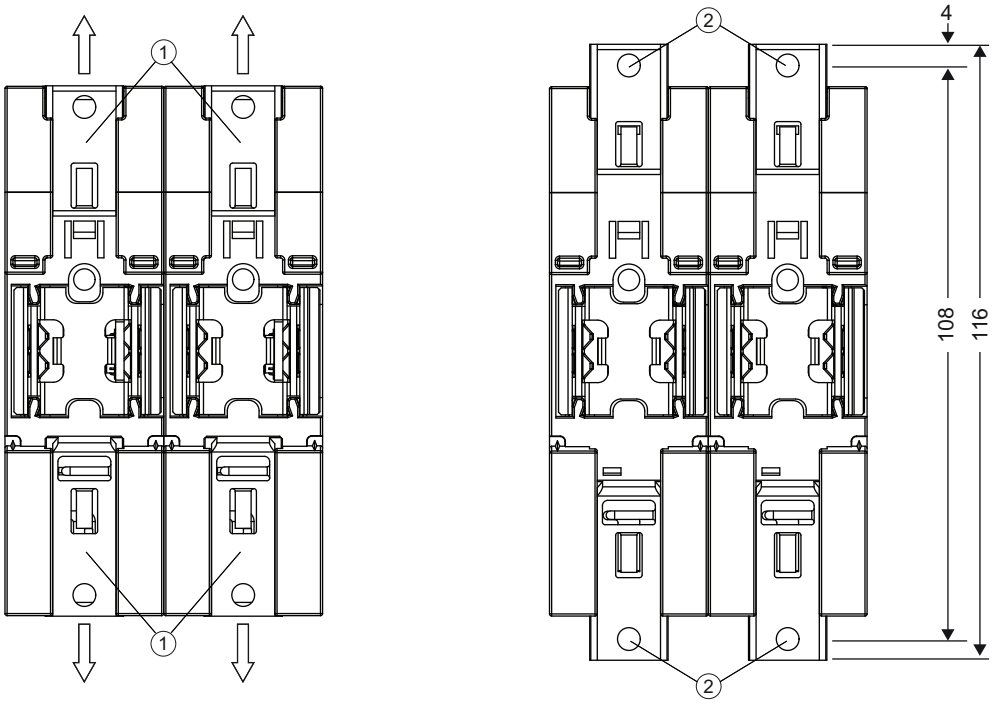
Before you remove power from the unit, disconnect the grounding of the TS Adapter by removing the analog cable and Ethernet cable.

3.3.8.4 Installing the TS adapter on a wall

Prerequisites: You must have connected the TS Adapter and TS module.

1. Move the attachment slider ① to the backside of the TS Adapter and TS module in the direction of the arrow until it engages.
2. Screw the TS Adapter and TS module to the position marked with ② to the designated assembly wall.

The following illustration shows the TS Adapter from behind, with the attachment sliders ① in both positions:



① Attachment slider
② Drill holes for wall mounting

3.4 Wiring guidelines

Proper grounding and wiring of all electrical equipment is important to help ensure the optimum operation of your system and to provide additional electrical noise protection for your application and the S7-1200. Refer to the technical specifications (Page 657) for the S7-1200 wiring diagrams.

Prerequisites

Before you ground or install wiring to any electrical device, ensure that the power to that equipment has been turned off. Also, ensure that the power to any related equipment has been turned off.

Ensure that you follow all applicable electrical codes when wiring the S7-1200 and related equipment. Install and operate all equipment according to all applicable national and local standards. Contact your local authorities to determine which codes and standards apply to your specific case.

⚠ WARNING

Installation or wiring the S7-1200 or related equipment with power applied could cause electric shock or unexpected operation of equipment. Failure to disable all power to the S7-1200 and related equipment during installation or removal procedures could result in death, severe personal injury, and/or damage due to electric shock or unexpected equipment operation.

Always follow appropriate safety precautions and ensure that power to the S7-1200 is disabled before attempting to install or remove the S7-1200 or related equipment.

Always take safety into consideration as you design the grounding and wiring of your S7-1200 system. Electronic control devices, such as the S7-1200, can fail and can cause unexpected operation of the equipment that is being controlled or monitored. For this reason, you should implement safeguards that are independent of the S7-1200 to protect against possible personal injury or equipment damage.

⚠ WARNING

Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death, severe personal injury and/or property damage.

Use an emergency stop function, electromechanical overrides, or other redundant safeguards that are independent of the S7-1200.

Guidelines for isolation

S7-1200 AC power supply boundaries and I/O boundaries to AC circuits have been designed and approved to provide safe separation between AC line voltages and low voltage circuits. These boundaries include double or reinforced insulation, or basic plus supplementary insulation, according to various standards. Components which cross these boundaries such as optical couplers, capacitors, transformers, and relays have been approved as providing safe separation. Isolation boundaries which meet these requirements have been identified in S7-1200 product data sheets as having 1500 VAC or greater isolation. This designation is based on a routine factory test of (2Ue + 1000 VAC) or equivalent according to approved methods. S7-1200 safe separation boundaries have been type tested to 4242 VDC.

The sensor supply output, communications circuits, and internal logic circuits of an S7-1200 with included AC power supply are sourced as SELV (safety extra-low voltage) according to EN 61131-2.

To maintain the safe character of the S7-1200 low voltage circuits, external connections to communications ports, analog circuits, and all 24 V nominal power supply and I/O circuits must be powered from approved sources that meet the requirements of SELV, PELV, Class 2, Limited Voltage, or Limited Power according to various standards.

⚠ WARNING

Use of non-isolated or single insulation supplies to supply low voltage circuits from an AC line can result in hazardous voltages appearing on circuits that are expected to be touch safe, such as communications circuits and low voltage sensor wiring.

Such unexpected high voltages could cause electric shock resulting in death, severe personal injury and/or property damage.

Only use high voltage to low voltage power converters that are approved as sources of touch safe, limited voltage circuits.

Guidelines for grounding the S7-1200

The best way to ground your application is to ensure that all the common and ground connections of your S7-1200 and related equipment are grounded to a single point. This single point should be connected directly to the earth ground for your system.

All ground wires should be as short as possible and should use a large wire size, such as 2 mm^2 (14 AWG).

When locating grounds, consider safety-grounding requirements and the proper operation of protective interrupting devices.

Guidelines for wiring the S7-1200

When designing the wiring for your S7-1200, provide a single disconnect switch that simultaneously removes power from the S7-1200 CPU power supply, from all input circuits, and from all output circuits. Provide over-current protection, such as a fuse or circuit breaker, to limit fault currents on supply wiring. Consider providing additional protection by placing a fuse or other current limit in each output circuit.

Install appropriate surge suppression devices for any wiring that could be subject to lightning surges.

Avoid placing low-voltage signal wires and communications cables in the same wire tray with AC wires and high-energy, rapidly switched DC wires. Always route wires in pairs, with the neutral or common wire paired with the hot or signal-carrying wire.

Use the shortest wire possible and ensure that the wire is sized properly to carry the required current. The CPU and SM connector accepts wire sizes from 2 mm^2 to 0.3 mm^2 (14 AWG to 22 AWG). The SB connector accepts wire sizes from 1.3 mm^2 to 0.3 mm^2 (16 AWG to 22 AWG). Use shielded wires for optimum protection against electrical noise. Typically, grounding the shield at the S7-1200 gives the best results.

When wiring input circuits that are powered by an external power supply, include an overcurrent protection device in that circuit. External protection is not necessary for circuits that are powered by the 24 VDC sensor supply from the S7-1200 because the sensor supply is already current-limited.

All S7-1200 modules have removable connectors for user wiring. To prevent loose connections, ensure that the connector is seated securely and that the wire is installed securely into the connector. To avoid damaging the connector, be careful that you do not over-tighten the screws. The maximum torque for the CPU and SM connector screw is 0.56 N-m (5 inch-pounds). The maximum torque for the SB connector screw is 0.33 N-m (3 inch-pounds).

To help prevent unwanted current flows in your installation, the S7-1200 provides isolation boundaries at certain points. When you plan the wiring for your system, you should consider these isolation boundaries. Refer to the technical specifications for the amount of isolation provided and the location of the isolation boundaries. Do not depend on isolation boundaries rated less than 1500 VAC as safety boundaries.

Guidelines for lamp loads

Lamp loads are damaging to relay contacts because of the high turn-on surge current. This surge current will nominally be 10 to 15 times the steady state current for a Tungsten lamp. A replaceable interposing relay or surge limiter is recommended for lamp loads that will be switched a large number of times during the lifetime of the application.

Guidelines for inductive loads

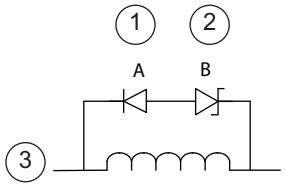
You should equip inductive loads with suppression circuits to limit voltage rise when the control output turns off. Suppression circuits protect your outputs from premature failure due to the high voltages associated with turning off inductive loads. In addition, suppression circuits limit the electrical noise generated when switching inductive loads. Placing an external suppression circuit so that it is electrically across the load, and physically located near the load is most effective in reducing electrical noise.

S7-1200 DC outputs include internal suppression circuits that are adequate for the inductive loads in most applications. Since S7-1200 relay output contacts can be used to switch either a DC or an AC load, internal protection is not provided.

Note

The effectiveness of a given suppression circuit depends on the application, and you must verify it for your particular use. Always ensure that all components used in your suppression circuit are rated for use in the application.

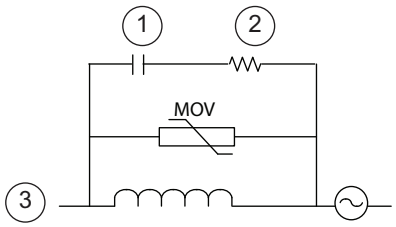
Typical suppressor circuit for DC or relay outputs that switch DC inductive loads



① 1N4001 diode or equivalent
② 8.2 V Zener (DC outputs), 36 V Zener (Relay outputs)
③ Output point

In most applications, the addition of a diode (A) across a DC inductive load is suitable, but if your application requires faster turn-off times, then the addition of a Zener diode (B) is recommended. Be sure to size your Zener diode properly for the amount of current in your output circuit.

Typical suppressor circuit for relay outputs that switch AC inductive loads



① 0.1 µ F
② 100 to 120 Ω
③ Output point

When you use a relay output to switch 115 V/230 VAC loads, place the appropriately rated resistor-capacitor-metal oxide varistor (MOV) circuit across the AC load. Ensure that the working voltage of the MOV is at least 20% greater than the nominal line voltage.

PLC concepts

4

4.1 Execution of the user program

The CPU supports the following types of code blocks that allow you to create an efficient structure for your user program:

- Organization blocks (OBs) define the structure of the program. Some OBs have predefined behavior and start events, but you can also create OBs with custom start events. Valid OB number ranges are shown in Event execution priorities and queuing (Page 68).
- Functions (FCs) and function blocks (FBs) contain the program code that corresponds to specific tasks or combinations of parameters. Each FC or FB provides a set of input and output parameters for sharing data with the calling block. An FB also uses an associated data block (called an instance DB) to maintain state of values between execution that can be used by other blocks in the program. Valid FC and FB numbers range from 1 to 65535.
- Data blocks (DBs) store data that can be used by the program blocks. Valid DB numbers range from 1 to 65535.

Execution of the user program begins with one or more optional start-up organization blocks (OBs) which are executed once upon entering RUN mode, followed by one or more program cycle OBs which are executed cyclically. An OB can also be associated with an interrupt event, which can be either a standard event or an error event, and executes whenever the corresponding standard or error event occurs.

A function (FC) or a function block (FB) is a block of program code that can be called from an OB or from another FC or FB, down to the following nesting depths:

- 16 from the program cycle or startup OB
- 4 from time delay interrupt, cyclic interrupt, time of day interrupt, hardware interrupt, time error interrupt, or diagnostic error interrupt OB

FCs are not associated with any particular data block (DB), while FBs are tied directly to a DB and use the DB for passing parameters and storing interim values and results.

The size of the user program, data, and configuration is limited by the available load memory and work memory in the CPU. There is no specific limit to the number of each individual OB, FC, FB and DB block. However, the total number of blocks is limited to 1024.

Each cycle includes writing the outputs, reading the inputs, executing the user program instructions, and performing background processing. The cycle is referred to as a scan cycle or scan.

The modules (SM, SB, CB, CM or CP) are detected and logged in only upon power-up.

- Inserting or removing a module in the central rack under power (hot) is not supported. Never insert or remove a module from the central rack when the CPU has power..

> **WARNING**
>
> Insertion or removal of a module (SM, SB, CD, CM or CP) from the central rack when the CPU has power could cause unpredictable behavior, resulting in damage to equipment and/or injury to personnel.
>
> Always ensure that power is removed from the CPU and central rack before inserting or removing a module from the central rack.

- You can insert or remove a SIMATIC memory card while the CPU is under power. However, inserting or removing a memory card when the CPU is in RUN mode causes the CPU to go to STOP mode.

> **CAUTION**
>
> Insertion or removal of a memory card when the CPU is in RUN mode causes the CPU to go to STOP, which might result in in damage to the equipment or the process being controlled.
>
> Whenever you insert or remove a memory card, the CPU immediately goes to STOP mode. Before inserting or removing a memory card, always ensure that the CPU is not actively controlling a machine or process. Always install an emergency stop circuit for your application or process.

- If you insert or remove a module in a distributed I/O rack (PROFINET or PROFIBUS) when the CPU is in RUN mode, the CPU generates an entry in the diagnostics buffer and stays in RUN mode.

Under the default configuration, all local digital and analog I/O points are updated synchronously with the scan cycle using an internal memory area called the process image. The process image contains a snapshot of the physical inputs and outputs (the physical I/O points on the CPU, signal board, and signal modules).

The CPU performs the following tasks:

- The CPU writes the outputs from the process image output area to the physical outputs.
- The CPU reads the physical inputs just prior to the execution of the user program and stores the input values in the process image input area. This ensures that these values remain consistent throughout the execution of the user instructions.
- The CPU executes the logic of the user instructions and updates the output values in the process image output area instead of writing to the actual physical outputs.

This process provides consistent logic through the execution of the user instructions for a given cycle and prevents the flickering of physical output points that might change state multiple times in the process image output area.

You can specify whether digital and analog I/O points are to be automatically updated and stored in the process image. If you insert a module in the device view, its data is located in the process image of the CPU (default). The CPU handles the data exchange between the module and the process image area automatically during the update of the process image. To remove digital or analog points from the process-image automatic update, select the appropriate device in Device configuration, view the Properties tab, expand if necessary to locate the desired I/O points, and then select "IO addresses/HW identifier". Then change the entry for "Process image:" from "Cyclic PI" to "---". To add the points back to the process-image automatic update, change this selection back to "Cyclic PI".

You can immediately read physical input values and immediately write physical output values when an instruction executes. An immediate read accesses the current state of the physical input and does not update the process image input area, regardless of whether the point is configured to be stored in the process image. An immediate write to the physical output updates both the process image output area (if the point is configured to be stored in the process image) and the physical output point. Append the suffix ":P" to the I/O address if you want the program to immediately access I/O data directly from the physical point instead of using the process image.

The CPU supports distributed I/O for both PROFINET and PROFIBUS networks (Page 393).

4.1.1 Operating modes of the CPU

The CPU has three modes of operation: STOP mode, STARTUP mode, and RUN mode. Status LEDs on the front of the CPU indicate the current mode of operation.

- In STOP mode, the CPU is not executing the program. You can download a project.
- In STARTUP mode, the startup OBs (if present) are executed once. Interrupt events are not processed during the startup mode.
- In RUN mode, the program cycle OBs are executed repeatedly. Interrupt events can occur and be processed at any point within the RUN mode. Some parts of a project can be downloaded in RUN mode (Page 650).

The CPU supports a warm restart for entering the RUN mode. Warm restart does not include a memory reset. All non-retentive system and user data are initialized at warm restart. Retentive user data is retained.

A memory reset clears all work memory, clears retentive and non-retentive memory areas, and copies load memory to work memory. A memory reset does not clear the diagnostics buffer or the permanently saved values of the IP address.

Note

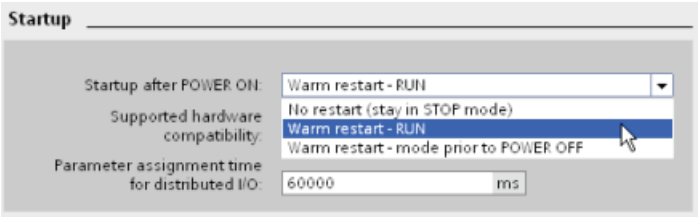
When you download one or more DBs from STEP 7 V11 to an S7-1200 V2 CPU, the retentive and non-retentive values of those DBs are set to their start values. The next transition to RUN performs a warm restart, setting all non-retentive data to their start values and setting all retentive data to their retained values.

When you download project elements (such as device configuration, code blocks or DBs) from STEP 7 V10.5 to any S7-1200 CPU or from STEP 7 V11 to an S7-1200 V1 CPU (or a V2 CPU that has been configured as a V1 CPU), the next transition to RUN mode resets **all** of the DBs in the project to their start values.

You can configure the "startup after POWER ON" setting of the CPU. This configuration item appears under the "Device configuration" for the CPU under "Startup". When power is applied, the CPU performs a sequence of power-up diagnostic checks and system initialization. During system initialization the CPU deletes all non-retentive bit memory and resets all non-retentive DB contents to the initial values from load memory. The CPU retains retentive bit memory and retentive DB contents and then enters the appropriate operating mode. Certain detected errors prevent the CPU from entering the RUN mode. The CPU supports the following configuration choices:

- No restart (stay in STOP mode)
- Warm restart - RUN
- Warm restart - mode prior to POWER OFF

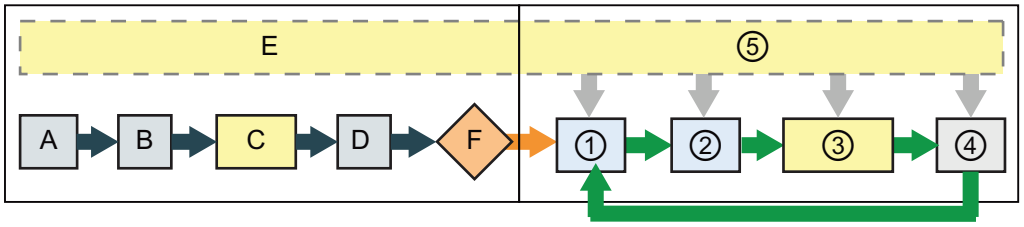
You can change the current operating mode using the "STOP" or "RUN" commands (Page 639) from the online tools of the programming software. You can also include a STP instruction (Page 222) in your program to change the CPU to STOP mode. This allows you to stop the execution of your program based on the program logic.



- In STOP mode, the CPU handles any communication requests (as appropriate) and performs self-diagnostics. The CPU does not execute the user program, and the automatic updates of the process image do not occur.

  You can download your project only when the CPU is in STOP mode.

- In STARTUP and RUN modes, the CPU performs the tasks shown in the following figure.



STARTUP

- A Clears the I (image) memory area
- B Initializes the outputs with either the last value or the substitute value
- C Executes the startup OBs
- D Copies the state of the physical inputs to I memory
- E Stores any interrupt events into the queue to be processed after entering RUN mode
- F Enables the writing of Q memory to the physical outputs

RUN

- ① Writes Q memory to the physical outputs
- ② Copies the state of the physical inputs to I memory
- ③ Executes the program cycle OBs
- ④ Performs self-test diagnostics
- ⑤ Processes interrupts and communications during any part of the scan cycle

STARTUP processing

Whenever the operating mode changes from STOP to RUN, the CPU clears the process image inputs, initializes the process image outputs and processes the startup OBs. Any read accesses to the process-image inputs by instructions in the startup OBs read zero rather than the current physical input value. Therefore, to read the current state of a physical input during the startup mode, you must perform an immediate read. The startup OBs and any associated FCs and FBs are executed next. If more than one startup OB exists, each is executed in order according to the OB number, with the lowest OB number executing first.

Each startup OB includes startup information that helps you determine the validity of retentive data and the time-of-day clock. You can program instructions inside the startup OBs to examine these startup values and to take appropriate action. The following startup locations are supported by the Startup OBs:

Table 4- 1 Startup locations supported by the startup OB

Input	Data Type	Description
LostRetentive	Bool	This bit is true if the retentive data storage areas have been lost
LostRTC	Bool	This bit is true if the time-of-day clock (Real time Clock) has been lost

The CPU also performs the following tasks during the startup processing.

- Interrupts are queued but not processed during the startup phase
- No cycle time monitoring is performed during the startup phase
- Configuration changes to HSC (high-speed counter), PWM (pulse-width modulation), and PtP (point-to-point communication) modules can be made in startup
- Actual operation of HSC, PWM and point-to-point communication modules only occurs in RUN

After the execution of the startup OBs finishes, the CPU goes to RUN mode and processes the control tasks in a continuous scan cycle.

4.1.2 Processing the scan cycle in RUN mode

For each scan cycle, the CPU writes the outputs, reads the inputs, executes the user program, updates communication modules, and responds to user interrupt events and communication requests. Communication requests are handled periodically throughout the scan.

These actions (except for user interrupt events) are serviced regularly and in sequential order. User interrupt events which are enabled are serviced according to priority in the order in which they occur.

The system guarantees that the scan cycle will be completed in a time period called the maximum cycle time; otherwise a time error event is generated.

- Each scan cycle begins by retrieving the current values of the digital and analog outputs from the process image and then writing them to the physical outputs of the CPU, SB, and SM modules configured for automatic I/O update (default configuration). When a physical output is accessed by an instruction, both the output process image and the physical output itself are updated.
- The scan cycle continues by reading the current values of the digital and analog inputs from the CPU, SB, and SMs configured for automatic I/O update (default configuration), and then writing these values to the process image. When a physical input is accessed by an instruction, the value of the physical input is accessed by the instruction, but the input process image is not updated.
- After reading the inputs, the user program is executed from the first instruction through the end instruction. This includes all the program cycle OBs plus all their associated FCs and FBs. The program cycle OBs are executed in order according to the OB number with the lowest OB number executing first.

Communications processing occurs periodically throughout the scan, possibly interrupting user program execution.

Self-diagnostic checks include periodic checks of the system and the I/O module status checks.

Interrupts can occur during any part of the scan cycle, and are event-driven. When an event occurs, the CPU interrupts the scan cycle and calls the OB that was configured to process that event. After the OB finishes processing the event, the CPU resumes execution of the user program at the point of interruption.

4.1.3 Organization blocks (OBs)

OBs control the execution of the user program. Each OB must have a unique OB number. The default OB numbers are reserved below 200. Other OBs must be numbered 200 or greater.

Specific events in the CPU trigger the execution of an organization block. OBs cannot call each other or be called from an FC or FB. Only a start event, such as a diagnostic interrupt or a time interval, can start the execution of an OB. The CPU handles OBs according to their respective priority classes, with higher priority OBs executed before lower priority OBs. The lowest priority class is 1 (for the main program cycle), and the highest priority class is 27 (for the time-error interrupts).

OBs control the following operations:

- Program cycle OBs execute cyclically while the CPU is in RUN mode. The main block of the program is a program cycle OB. This is where you place the instructions that control your program and where you call additional user blocks. Multiple program cycle OBs are allowed and are executed in numerical order. OB 1 is the default. Other program cycle OBs must be identified as OB 200 or greater.
- Startup OBs execute one time when the operating mode of the CPU changes from STOP to RUN, including powering up in the RUN mode and in commanded STOP-to-RUN transitions. After completion, the main "Program cycle" OB will begin executing. Multiple startup OBs are allowed. OB 100 is the default. Others must be OB 200 or greater.
- Cyclic interrupt OBs execute at a specified interval. A cyclic interrupt OB will interrupt cyclic program execution at user defined intervals, such as every 2 seconds. You can configure up to a total of 4 for both the time-delay and cyclic events at any given time, with one OB allowed for each configured time-delay and cyclic event. The OB must be OB 200 or greater.
- Hardware interrupt OBs execute when the relevant hardware event occurs, including rising and falling edges on built-in digital inputs and HSC events. A hardware interrupt OB will interrupt normal cyclic program execution in reaction to a signal from a hardware event. You define the events in the properties of the hardware configuration. One OB is allowed for each configured hardware event. The OB must be OB 200 or greater.
- A time error interrupt OB executes when either the maximum cycle time is exceeded or a time error event occurs. The OB for processing the time error interrupt is OB 80. If triggered, it executes, interrupting normal cyclic program execution or any other event OB. The events that trigger the time error interrupt and the reaction of the CPU to those events are described below:
  - Exceeding the maximum cycle time: You configure the maximum cycle time in the properties of the CPU. If OB 80 does not exist, the reaction of the CPU for exceeding the maximum time is to change to STOP.
  - Time errors: If OB 80 does not exist, the reaction of the CPU is to stay in RUN. Time errors occur if the time of day event is missed or repeated, the queue overflows, or an event OB (time delay event, time of day event, or cyclic interrupt) starts before the CPU finishes the execution of the first.

  The occurrence of either of these events generates a diagnostic buffer entry describing the event. The diagnostic buffer entry is generated regardless of the existence of OB 80.
- Diagnostic error interrupt OBs execute when a diagnostic error is detected and reported. A diagnostic OB interrupts the normal cyclic program execution if a diagnostics-capable module recognizes an error (if the diagnostic error interrupt has been enabled for the module). OB 82 is the only OB number supported for the diagnostic error event. You can include an STP instruction (put CPU in STOP mode) inside your OB 82 if you desire your CPU to enter STOP mode upon receiving this type of error. If there is no diagnostic OB in the program, the CPU ignores the error (stays in RUN).

4.1.4 Event execution priorities and queuing

The CPU processing is controlled by events. An event triggers an interrupt OB to be executed. You can specify the interrupt OB for an event during the creation of the block, during the device configuration, or with an ATTACH or DETACH instruction. Some events happen on a regular basis like the program cycle or cyclic events. Other events happen only a single time, like the startup event and time delay events. Some events happen when there is a change triggered by the hardware, such as an edge event on an input point or a high speed counter event. There are also events like the diagnostic error and time error event which only happen when there is an error. The event priorities and queues are used to determine the processing order for the event interrupt OBs.

The program cycle event happens once during each program cycle (or scan). During the program cycle, the CPU writes the outputs, reads the inputs and executes program cycle OBs. The program cycle event is required and is always enabled. You may have no program cycle OBs, or you may have multiple OBs selected for the program cycle event. After the program cycle event is triggered, the lowest numbered program cycle OB (usually OB 1) is executed. The other program cycle OBs are executed sequentially (in numerical order) within the program cycle.

The cyclic interrupt events allow you to configure the execution of an interrupt OB at a configured scan time. The initial scan time is configured when the OB is created and selected to be a cyclic interrupt OB. A cyclic event will interrupt the program cycle and execute the cyclic interrupt OB (the cyclic event is at a higher priority class than the program cycle event).

Only one cyclic interrupt OB can be attached to a cyclic event.

Each cyclic event can be assigned a phase shift so that the execution of cyclic interrupts with the same scan time can be offset from one another by the phase shift amount. The default phase shift is 0. To change the initial phase shift, or to change the initial scan time for a cyclic event, right click on the cyclic interrupt OB in the project tree, click "Properties", then click "Cyclic interrupt", and enter the new initial values. You can also query and change the scan time and the phase shift from your program using the Query cyclic interrupt (QRY_CINT) and Set cyclic interrupt (SET_CINT) instructions. Scan time and phase shift values set by the SET_CINT instruction do not persist through a power cycle or a transition to STOP mode; scan time and phase shift values will return to the initial values following a power cycle or a transition to STOP. The CPU supports a total of four cyclic and time-delay interrupt events.

The startup event happens one time on a STOP to RUN transition and causes the startup OBs to be executed. Multiple OBs can be selected for the startup event. The startup OBs are executed in numerical order.

The time delay interrupt events allow you to configure the execution of an interrupt OB after a specified delay time has expired. The delay time is specified with the SRT_DINT instruction. The time delay events will interrupt the program cycle to execute the time delay interrupt OB. Only one time delay interrupt OB can be attached to a time delay event. The CPU supports four time delay events.

The hardware interrupt events are triggered by a change in the hardware, such as a rising or falling edge on an input point, or a HSC (High Speed Counter) event. There can be one interrupt OB selected for each hardware interrupt event. The hardware events are enabled in Device configuration. The OBs are specified for the event in the Device configuration or with an ATTACH instruction in the user program. The CPU supports several hardware interrupt events. The exact events are based on the CPU model and the number of input points.

The time and diagnostic error interrupt events are triggered when the CPU detects an error. These events are at a higher priority class that the other interrupt events and can interrupt the execution of the time delay, cyclic and hardware interrupt events. One interrupt OB can be specified for each of the time error and diagnostic error interrupt events.

Understanding event execution priorities and queuing

The number of pending (queued) events from a single source is limited, using a different queue for each event type. Upon reaching the limit of pending events for a given event type, the next event is lost. Refer to the following section on "Understanding time error events" for more information regarding queue overflows.

Each CPU event has an associated priority. You cannot change the priority of an OB. In general, events are serviced in order of priority (highest priority first). Events of the same priority are serviced on a "first-come, first-served" basis.

Table 4- 2 OB events

Event	OB number	Quantity allowed	Start event	OB priority
Program cycle	OB 1, OB 200 to OB 65535	1 program cycle event Multiple OBs allowed	• Startup OB ends • Last program cycle OB ends	1
Startup	OB 100, OB 200 to OB 65535	1 startup event [1, 2] Multiple OBs allowed	STOP-to-RUN transition	1
Time	OB 200 to OB 65535	Up to 4 time events[3] 1 OB per event	Time-delay OB event is scheduled	3
			Cyclic OB event is scheduled	4
Process	OB 200 to OB 65535	Up to 50 process events[4] 1 OB per event	Edges: • Rising edge events: 16 max. • Falling edge events: 16 max.	5
			For HSC: • CV=PV: 6 max. • Direction changed: 6 max. • External reset: 6 max.	6

Event	OB number	Quantity allowed	Start event	OB priority
Diagnostic error	OB 82	1 event (only if OB 82 was loaded)	Module transmits an error	9
Time error	OB 80	1 event (only if OB 80 was loaded)[5]	• Maximum cycle time was exceeded • A second time interrupt (cyclic or time-delay) started before the CPU had finished execution of the first interrupt	26

1 The startup event and the program cycle event will never occur at the same time because the startup event will run to completion before the program cycle event will be started (controlled by the operating system).

2 Only the diagnostic error event (OB 82) interrupts the startup event. All other events are queued to be processed after the startup event has finished.

3 The CPU provides a total of 4 time events that are shared by the time-delay OBs and the cyclic OBs. The number of time-delay and cyclic OBs in your user program cannot exceed 4.

4 You can have more than 50 process events if you use the DETACH and ATTACH instructions.

5 You can configure the CPU to stay in RUN if the maximum scan cycle time was exceeded or you can use the RE_TRIGR instruction to reset the cycle time. However, the CPU goes to STOP mode the second time that the maximum scan cycle time was exceeded in one scan cycle.

After the execution of an OB with a priority of 2 to 25 has started, processing of that OB cannot be interrupted by the occurrence of another event, except for by OB 80 (time-error event, which has a priority of 26). All other events are queued for later processing, allowing the current OB to finish.

Interrupt latency

The interrupt event latency (the time from notification of the CPU that an event has occurred until the CPU begins execution of the first instruction in the OB that services the event) is approximately 175 µsec, provided that a program cycle OB is the only event service routine active at the time of the interrupt event.

Understanding time error events

The occurrence of any of several different time error conditions results in a time error event. The following time errors are supported:

- Maximum cycle time exceeded
- Requested OB cannot be started
- Queue overflow occurred

The maximum cycle time exceeded condition results if the program cycle does not complete within the specified maximum scan cycle time. See the section on "Monitoring the cycle time" (Page 73) for more information regarding the maximum cycle time condition, how to configure the maximum scan cycle time, and how to reset the cycle timer.

The requested OB cannot be started condition results if an OB is requested by a cyclic interrupt, a time-delay interrupt, or a time-of-day interrupt, but the requested OB is already being executed.

The queue overflow occurred condition results if the interrupts are occurring faster than they can be processed. The number of pending (queued) events is limited using a different queue for each event type. If an event occurs when the corresponding queue is full, a time error event is generated.

All time error events trigger the execution of OB 80 if it exists. If an OB 80 is not included in the user program, then the device configuration of the CPU determines the CPU reaction to the time error:

- The default configuration for time errors, such as starting a second cyclic interrupt before the CPU has finished the execution of the first, is for the CPU to stay in RUN.
- The default configuration for exceeding the maximum time is for the CPU to change to STOP.

You can use the RE_TRIGR instruction to reset the maximum cycle time. However, if two "maximum cycle time exceeded" conditions occur within the same program cycle without resetting the cycle timer, then the CPU transitions to STOP, regardless of whether OB 80 exists. See the section on "Monitoring the cycle time" (Page 73).

OB 80 includes startup information that helps you determine which event and OB generated the time error. You can program instructions inside OB 80 to examine these startup values and to take appropriate action.

Table 4- 3 Startup information for OB 80

Input	Data type	Description
fault_id	BYTE	16#01 - maximum cycle time exceeded 16#02 - requested OB cannot be started 16#07 and 16#09 - queue overflow occurred
csg_OBnr	OB_ANY	Number of the OB which was being executed when the error occurred
csg_prio	UINT	Priority of the OB causing the error

No time error interrupt OB 80 is present when you create a new project. If desired, you add a time error interrupt OB 80 to your project by double-clicking "Add new block" under "Program blocks" in the tree, then choose "Organization block", and then "Time error interrupt".

Understanding diagnostic error events

Analog (local), PROFINET, and PROFIBUS devices are capable of detecting and reporting diagnostic errors. The occurrence or removal of any of several different diagnostic error conditions results in a diagnostic error event. The following diagnostic errors are supported:

- No user power
- High limit exceeded
- Low limit exceeded
- Wire break
- Short circuit

Diagnostic error events trigger the execution of OB 82 if it exists. If OB 82 does not exist, then the CPU ignores the error. No diagnostic error interrupt OB 82 is present when you create a new project. If desired, you add a diagnostic error interrupt OB 82 to your project by double-clicking "Add new block" under "Program blocks" in the tree, then choose "Organization block", and then "Diagnostic error interrupt".

Note

Diagnostic errors for multi-channel local analog devices (I/O, RTD, and Thermocouple)

The OB 82 diagnostic error interrupt can report only one channel's diagnostic error at a time.

If two channels of a multi-channel device have an error, then the second error only triggers OB 82 under the following conditions: the first channel error clears, the execution of OB 82 triggered by the first error is complete, and the second error still exists.

OB 82 includes startup information that helps you determine whether the event is due to the occurrence or removal of an error, and the device and channel which reported the error. You can program instructions inside OB 82 to examine these startup values and to take appropriate action.

Table 4- 4 Startup information for OB 82

Input	Data type	Description
IOstate	WORD	IO state of the device: • Bit 0 = 1 if the configuration is correct, and = 0 if the configuration is no longer correct. • Bit 4 = 1 if an error is present (such as a wire break). (Bit 4 = 0 if there is no error.) • Bit 5 = 1 if the configuration is **not** correct, and = 0 if the configuration is correct again. • Bit 6 = 1 if an I/O access error has occurred. Refer to laddr for the hardware identifier of the I/O with the access error. (Bit 6 = 0 if there is no error.)
laddr	HW_ANY	Hardware identifier of the device or functional unit that reported the error[1]
channel	UINT	Channel number
multierror	BOOL	TRUE if more than one error is present

[1] The laddr input contains the hardware identifier of the device or functional unit which returned the error. The hardware identifier is assigned automatically when components are inserted in the device or network view and appears in the Constants tab of PLC tags. A name is also assigned automatically for the hardware identifier. These entries in the Constants tab of the PLC tags cannot be changed.

4.1.5 Monitoring the cycle time

The cycle time is the time that the CPU operating system requires to execute the cyclic phase of the RUN mode. The CPU provides two methods of monitoring the cycle time:

- Maximum scan cycle time
- Fixed minimum scan cycle time

Scan cycle monitoring begins after the startup event is complete. Configuration for this feature appears under the "Device Configuration" for the CPU under "Cycle time".

The CPU always monitors the scan cycle and reacts if the maximum scan cycle time is exceeded. If the configured maximum scan cycle time is exceeded, an error is generated and is handled one of two ways:

- If the user program does not include an OB 80, then the CPU generates an error and goes to STOP. (You can change the configuration of the CPU to ignore this time error and stay in RUN. The default configuration is for the CPU to go to STOP.)
- If the user program includes an OB 80, then the CPU executes OB 80

The RE_TRIGR instruction (Re-trigger cycle time monitoring) allows you to reset the timer that measures the cycle time. However, this instruction only functions if executed in a program cycle OB; the RE_TRIGR instruction is ignored if executed in OB 80. If the maximum scan cycle time is exceeded twice within the same program cycle with no RE_TRIGR instruction execution between the two, then the CPU transitions to STOP immediately. The use of repeated executions of the RE_TRIGR instruction can create an endless loop or a very long scan.

Typically, the scan cycle executes as fast as it can be executed and the next scan cycle begins as soon as the current one completes. Depending upon the user program and communication tasks, the time period for a scan cycle can vary from scan to scan. To eliminate this variation, the CPU supports an optional fixed minimum scan cycle time (also called fixed scan cycle). When this optional feature is enabled and a fixed minimum scan cycle time is provided in ms, the CPU will maintain the minimum cycle time within ±1 ms for the completion of each CPU scan.

In the event that the CPU completes the normal scan cycle in less time than the specified minimum cycle time, the CPU spends the additional time of the scan cycle performing runtime diagnostics and/or processing communication requests. In this way the CPU always takes a fixed amount of time to complete a scan cycle.

In the event that the CPU does not complete the scan cycle in the specified minimum cycle time, the CPU completes the scan normally (including communication processing) and does not create any system reaction as a result of exceeding the minimum scan time. The following table defines the ranges and defaults for the cycle time monitoring functions.

Table 4- 5 Range for the cycle time

Cycle time	Range (ms)	Default
Maximum scan cycle time[1]	1 to 6000	150 ms
Fixed minimum scan cycle time[2]	1 to maximum scan cycle time	Disabled

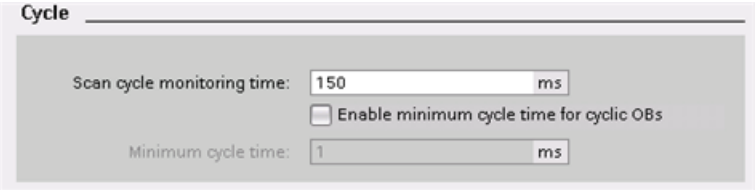
1 The maximum scan cycle time is always enabled. Configure a cycle time between 1 ms to 6000 ms. The default is 150 ms.

2 The fixed minimum scan cycle time is optional, and is disabled by default. If required, configure a cycle time between 1 ms and the maximum scan cycle time.

Configuring the cycle time and communication load

You use the CPU properties in the Device configuration to configure the following parameters:

- Cycle time: You can enter a maximum scan cycle time. You can also enter a fixed minimum scan cycle time.



- Communications load: You can configure a percentage of the time to be dedicated for communication tasks.

For more information about the scan cycle, see "Monitoring the cycle time". (Page 73)

4.1.6 CPU memory

Memory management

The CPU provides the following memory areas to store the user program, data, and configuration:

- Load memory is non-volatile storage for the user program, data and configuration. When a project is downloaded to the CPU, it is first stored in the Load memory area. This area is located either in a memory card (if present) or in the CPU. This non-volatile memory area is maintained through a power loss. The memory card supports a larger storage space than that built-in to the CPU.
- Work memory is volatile storage for some elements of the user project while executing the user program. The CPU copies some elements of the project from load memory into work memory. This volatile area is lost when power is removed, and is restored by the CPU when power is restored.
- Retentive memory is non-volatile storage for a limited quantity of work memory values. The retentive memory area is used to store the values of selected user memory locations during power loss. When a power down or power loss occurs, the CPU restores these retentive values upon power up.

To display the memory usage for the current project, right-click the CPU (or one of its blocks) and select "Resources" from the context. To display the memory usage for the current CPU, double-click "Online and diagnostics", expand "Diagnostics", and select "Memory".

Retentive memory

Data loss after power failure can be avoided by marking certain data as retentive. The following data can be configured to be retentive:

- Bit memory(M): You can define the precise width of the memory for bit memory in the PLC tag table or in the assignment list. Retentive bit memory always starts at MB0 and runs consecutively up through a specified number of bytes. Specify this value from the PLC tag table or in the assignment list by clicking the "Retain" toolbar icon. Enter the number of M bytes to retain starting at MB0.
- Tags of a function block (FB): If an FB was created with "Optimzed" selected, then the interface editor for this FB includes a "Retain" column. In this column, you can select either "Retentive", "Non-retentive", or "Set in IDB" individually for each tag. An instance DB that was created when this FB is placed in the program editor shows this retain column as well. You can only change the retentive state of a tag from within the instance DB interface editor if you selected "Set in IDB" (Set in instance data block) in the Retain selection for the tag in the optimized FB.

  If an FB was created with "Standard - compatible with S7-300/400" selected, then the interface editor for this FB does not include a "Retain" column. An instance DB created when this FB is inserted in the program editor shows a "Retain" column which is available for edit. In this case, selecting the "Retain" option for any tag results in **all** tags being selected. Similarly, deselecting the option for any tag results in **all** tags being deselected. For an FB that was configured to be "Standard - compatible with S7-300/400", you can change the retentive state from within the instance DB editor, but all tags are set to the same retentive state together.

  After you create the FB, you cannot change the option for "Standard - compatible with S7-300/400". You can only select this option when you create the FB. To determine whether an existing FB was configured for "Optimized" or "Standard - compatible with S7-300/400", right-click the FB in the Project tree, select "Properties", and then select "Attributes". The check box "Optimized block access" when selected shows you whether a block is optimized. Otherwise, it is standard and compatible with S7-300/400 CPUs.
- Tags of a global data block: The behavior of a global DB with regard to retentive state assignment is similar to that of an FB. Depending on the block access setting you can define the retentive state either for individual tags or for all tags of a global data block.
  - If you select "Optimized" when you create the DB, you can set the retentive state for each individual tag.
  - If you select "Standard - compatible with S7-300/400" when you create the DB, the retentive-state setting applies to all tags of the DB; either all tags are retentive or no tag is retentive.

A total of 2048 bytes of data can be retentive. To see how much is available, from the PLC tag table or the assignment list, click on the "Retain" toolbar icon. Although this is where the retentive range is specified for M memory, the second row indicates the total remaining memory available for M and DB combined. Note that for this value to be accurate, you must compile all data blocks with retentive tags.

Diagnostics buffer

The CPU supports a diagnostic buffer which contains an entry for each diagnostic event. Each entry includes a date and time the event occurred, an event category, and an event description. The entries are displayed in chronological order with the most recent event at the top. While the CPU maintains power, up to 50 most recent events are available in this log. When the log is full, a new event replaces the oldest event in the log. When power is lost, the ten most recent events are saved.

The following types of events are recorded in the diagnostics buffer:

- Each system diagnostic event; for example, CPU errors and module errors
- Each state change of the CPU (each power up, each transition to STOP, each transition to RUN)

To access the diagnostic buffer, you must be online. Locate the log under "Online & diagnostics / Diagnostics / Diagnostics buffer". For more information regarding troubleshooting and debugging, refer to the "Online and diagnostics" chapter (Page 640).

Time of day clock

The CPU supports a time-of-day clock. A super-capacitor supplies the energy required to keep the clock running during times when the CPU is powered down. The super-capacitor charges while the CPU has power. After the CPU has been powered up at least 24 hours, then the super-capacitor has sufficient charge to keep the clock running for typically 10 days.

The Time of Day Clock is set to system time, which is Coordinated Universal Time (UTC). STEP 7 sets the Time of Day Clock to system time. There are instructions to read the system time (RD_SYS_T) or local time (RD_LOC_T). Local time is calculated by using the time zone and daylight saving time offsets that you set in the CPU Clock device configuration.

Configure the time-of-day clock for the CPU under the "Time of day" property. You can also enable daylight saving time and specify the start and end times for daylight saving time. To set the time-of-day clock, you must be online and in the "Online & diagnostics" view of the CPU. Use the "Set time of day" function.

4.1.6.1 System and clock memory

You use the CPU properties to enable bytes for "system memory" and "clock memory". Your program logic can reference the individual bits of these functions by their tag names.

- You can assign one byte in M memory for system memory. The byte of system memory provides the following four bits that can be referenced by your user program by the following tag names:
  - First cycle: (Tag name "FirstScan") bit is set to1 for the duration of the first scan after the startup OB finishes. (After the execution of the first scan, the "first scan" bit is set to 0.)
  - Diagnostics status changed (Tag name: "DiagStatusUpdate") is set to 1 for one scan after the CPU logs a diagnostic event. Because the CPU does not set the "diagnostic graph changed" bit until the end of the first execution of the program cycle OBs, your user program cannot detect if there has been a diagnostic change either during the execution of the startup OBs or the first execution of the program cycle OBs.
  - Always 1 (high): (Tag name "AlwaysTRUE") bit is always set to 1.
  - Always 0 (low): (Tag name "AlwaysFALSE") bit is always set to 0.
- You can assign one byte in M memory for clock memory. Each bit of the byte configured as clock memory generates a square wave pulse. The byte of clock memory provides 8 different frequencies, from 0.5 Hz (slow) to 10 Hz (fast). You can use these bits as control bits, especially when combined with edge instructions, to trigger actions in the user program on a cyclic basis.

The CPU initializes these bytes on the transition from STOP mode to STARTUP mode. The bits of the clock memory change synchronously to the CPU clock throughout the STARTUP and RUN modes.

⚠ CAUTION

Overwriting the system memory or clock memory bits can corrupt the data in these functions and cause your user program to operate incorrectly, which can cause damage to equipment and injury to personnel.

Because both the clock memory and system memory are unreserved in M memory, instructions or communications can write to these locations and corrupt the data.

Avoid writing data to these locations to ensure the proper operation of these functions, and always implement an emergency stop circuit for your process or machine.

System memory configures a byte with bits that turn on (value = 1) for a specific event.

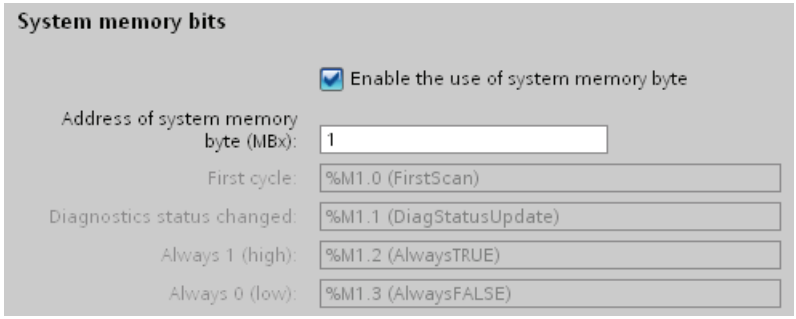


Table 4- 6 System memory

7	6	5	4	3	2	1	0
Reserved Value 0				Always off Value 0	Always on Value 1	Diagnostic status indicator • 1: Change • 0: No change	First scan indicator • 1: First scan after startup • 0: Not first scan

Clock memory configures a byte that cycles the individual bits on and off at fixed intervals. Each clock bit generates a square wave pulse on the corresponding M memory bit. These bits can be used as control bits, especially when combined with edge instructions, to trigger actions in the user code on a cyclic basis.

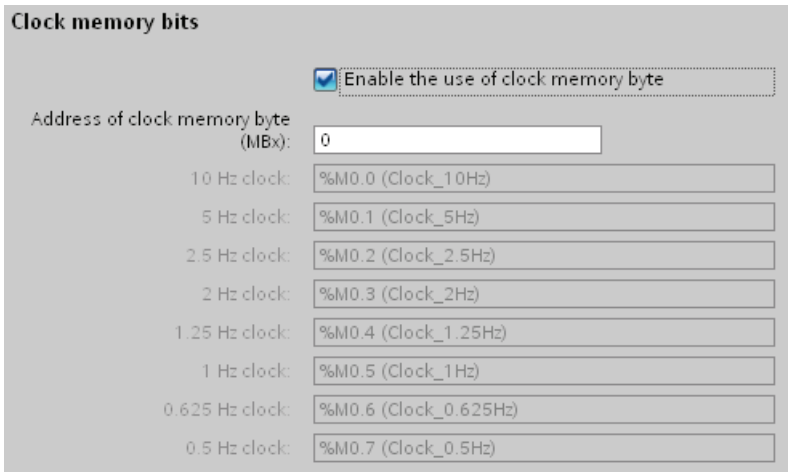


Table 4- 7 Clock memory

Bit number	7	6	5	4	3	2	1	0
Tag name								
Period (s)	2.0	1.6	1.0	0.8	0.5	0.4	0.2	0.1
Frequency (Hz)	0.5	0.625	1	1.25	2	2.5	5	10

Because clock memory runs asynchronously to the CPU cycle, the status of the clock memory can change several times during a long cycle.

4.1.6.2 Configuring the outputs on a RUN-to-STOP transition

You can configure the behavior of the digital and analog outputs when the CPU is in STOP mode. For any output of a CPU, SB or SM, you can set the outputs to either freeze the value or use a substitute value:

- Substituting a specified output value (default): You enter a substitute value for each output (channel) of that CPU, SB, or SM device.

  The default substitute value for digital output channels is OFF, and the default substitute value for analog output channels is 0.

- Freezing the outputs to remain in last state: The outputs retain their current value at the time of the transition from RUN to STOP. After power up, the outputs are set to the default substitute value.

You configure the behavior of the outputs in Device Configuration. Select the individual devices and use the "Properties" tab to configure the outputs for each device.

When the CPU changes from RUN to STOP, the CPU retains the process image and writes the appropriate values for both the digital and analog outputs, based upon the configuration.

4.2 Data storage, memory areas, I/O and addressing

4.2.1 Accessing the data of the S7-1200

STEP 7 facilitates symbolic programming. You create symbolic names or "tags" for the addresses of the data, whether as PLC tags relating to memory addresses and I/O points or as local variables used within a code block. To use these tags in your user program, simply enter the tag name for the instruction parameter.

For a better understanding of how the CPU structures and addresses the memory areas, the following paragraphs explain the "absolute" addressing that is referenced by the PLC tags. The CPU provides several options for storing data during the execution of the user program:

- Global memory: The CPU provides a variety of specialized memory areas, including inputs (I), outputs (Q) and bit memory (M). This memory is accessible by all code blocks without restriction
- PLC tag table: You can enter symbolic names in the STEP 7 PLC tag table for specific memory locations. These tags are global to the STEP 7 program and allow programming with names that are meaningful for your application.
- Data block (DB): You can include DBs in your user program to store data for the code blocks. The data stored persists when the execution of the associated code block comes to an end. A "global" DB stores data that can be used by all code blocks, while an instance DB stores data for a specific FB and is structured by the parameters for the FB.
- Temp memory: Whenever a code block is called, the operating system of the CPU allocates the temporary, or local, memory (L) to be used during the execution of the block. When the execution of the code block finishes, the CPU reallocates the local memory for the execution of other code blocks.

Each different memory location has a unique address. Your user program uses these addresses to access the information in the memory location. References to the input (I) or output (Q) memory areas, such as I0.3 or Q1.7, access the process image. To immediately access the physical input or output, append the reference with ":P" (such as I0.3:P, Q1.7:P, or "Stop:P").

Table 4- 8 Memory areas

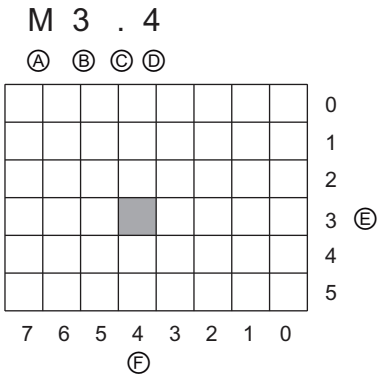
Memory area	Description	Force	Retentive
I Process image input	Copied from physical inputs at the beginning of the scan cycle	No	No
I_:P [1] (Physical input)	Immediate read of the physical input points on the CPU, SB, and SM	Yes	No
Q Process image output	Copied to physical outputs at the beginning of the scan cycle	No	No
Q_:P [1] (Physical output)	Immediate write to the physical output points on the CPU, SB, and SM	Yes	No
M Bit memory	Control and data memory	No	Yes (optional)
L Temp memory	Temporary data for a block local to that block	No	No
DB Data block	Data memory and also parameter memory for FBs	No	Yes (optional)

[1] To immediately access (read or write) the physical inputs and physical outputs, append a ":P" to the address or tag (such as I0.3:P, Q1.7:P, or "Stop:P").

Each different memory location has a unique address. Your user program uses these addresses to access the information in the memory location. The absolute address consists of the following elements:

- Memory area identifier (such as I, Q, or M)
- Size of the data to be accessed ("B' for Byte, "W" for Word, or "D" for DWord)
- Starting address of the data (such as byte 3 or word 3)

When accessing a bit in the address for a Boolean value, you do not enter a mnemonic for the size. You enter only the memory area, the byte location, and the bit location for the data (such as I0.0, Q0.1, or M3.4).



A	Memory area identifier	E	Bytes of the memory area
B	Byte address: byte 3	F	Bits of the selected byte
C	Separator ("byte.bit")		
D	Bit location of the byte (bit 4 of 8)		

In the example, the memory area and byte address (M = bit memory area, and 3 = Byte 3) are followed by a period (".") to separate the bit address (bit 4).

Accessing the data in the memory areas of the CPU

STEP 7 facilitates symbolic programming. Typically, tags are created either in PLC tags, a data block, or in the interface at the top of an OB, FC, or FB. These tags include a name, data type, offset, and comment. Additionally, in a data block, a start value can be specified. You can use these tags when programming by entering the tag name at the instruction parameter. Optionally you can enter the absolute operand (memory area, size and offset) at the instruction parameter. The examples in the following sections show how to enter absolute operands. The % character is inserted automatically in front of the absolute operand by the program editor. You can toggle the view in the program editor to one of these: symbolic, symbolic and absolute, or absolute.

I (process image input): The CPU samples the peripheral (physical) input points just prior to the cyclic OB execution of each scan cycle and writes these values to the input process image. You can access the input process image as bits, bytes, words, or double words. Both read and write access is permitted, but typically, process image inputs are only read.

Table 4- 9 Absolute addressing for I memory

Bit	I[byte address].[bit address]	I0.1
Byte, Word, or Double Word	I[size][starting byte address]	IB4, IW5, or ID12

By appending a ":P" to the address, you can immediately read the digital and analog inputs of the CPU, SB or SM. The difference between an access using I_:P instead of I is that the data comes directly from the points being accessed rather than from the input process image. This I_:P access is referred to as an "immediate read" access because the data is retrieved immediately from the source instead of from a copy that was made the last time the input process image was updated.

Because the physical input points receive their values directly from the field devices connected to these points, writing to these points is prohibited. That is, I_:P accesses are read-only, as opposed to I accesses which can be read or write.

I_:P accesses are also restricted to the size of inputs supported by a single CPU, SB, or SM, rounded up to the nearest byte. For example, if the inputs of a 2 DI / 2 DQ SB are configured to start at I4.0, then the input points can be accessed as I4.0:P and I4.1:P or as IB4:P. Accesses to I4.2:P through I4.7:P are not rejected, but make no sense since these points are not used. Accesses to IW4:P and ID4:P are prohibited since they exceed the byte offset associated with the SB.

Accesses using I_:P do not affect the corresponding value stored in the input process image.

Table 4- 10 Absolute addressing for I memory (immediate)

Bit	I[byte address].[bit address]:P	I0.1:P
Byte, Word, or Double word	I[size][starting byte address]:P	IB4:P, IW5:P, or ID12:P

Q (process image output): The CPU copies the values stored in the output process image to the physical output points. You can access the output process image in bits, bytes, words, or double words. Both read and write access is permitted for process image outputs.

Table 4- 11 Absolute addressing for Q memory

Bit	Q[byte address].[bit address]	Q1.1
Byte, Word, or Double word	Q[size][starting byte address]	QB5, QW10, QD40

By appending a ":P" to the address, you can immediately write to the physical digital and analog outputs of the CPU, SB or SM. The difference between an access using Q_:P instead of Q is that the data goes directly to the points being accessed in addition to the output process image (writes to both places). This Q_:P access is sometimes referred to as an "immediate write" access because the data is sent immediately to the target point; the target point does not have to wait for the next update from the output process image.

Because the physical output points directly control field devices that are connected to these points, reading from these points is prohibited. That is, Q_:P accesses are write-only, as opposed to Q accesses which can be read or write.

Q_:P accesses are also restricted to the size of outputs supported by a single CPU, SB, or SM, rounded up to the nearest byte. For example, if the outputs of a 2 DI / 2 DQ SB are configured to start at Q4.0, then the output points can be accessed as Q4.0:P and Q4.1:P or as QB4:P. Accesses to Q4.2:P through Q4.7:P are not rejected, but make no sense since these points are not used. Accesses to QW4:P and QD4:P are prohibited since they exceed the byte offset associated with the SB.

Accesses using Q_:P affect both the physical output as well as the corresponding value stored in the output process image.

Table 4- 12 Absolute addressing for Q memory (immediate)

Bit	Q[byte address].[bit address]:P	Q1.1:P
Byte, Word, or Double word	Q[size][starting byte address]:P	QB5:P, QW10:P or QD40:P

M (bit memory area): Use the bit memory area (M memory) for both control relays and data to store the intermediate status of an operation or other control information. You can access the bit memory area in bits, bytes, words, or double words. Both read and write access is permitted for M memory.

Table 4- 13 Absolute addressing for M memory

Bit	M[byte address].[bit address]	M26.7
Byte, Word, or Double Word	M[size][starting byte address]	MB20, MW30, MD50

Temp (temporary memory): The CPU allocates the temp memory on an as-needed basis. The CPU allocates the temp memory for the code block at the time when the code block is started (for an OB) or is called (for an FC or FB). The allocation of temp memory for a code block might reuse the same temp memory locations previously used by a different OB, FC or FB. The CPU does not initialize the temp memory at the time of allocation and therefore the temp memory might contain any value.

Temp memory is similar to M memory with one major exception: M memory has a "global" scope, and temp memory has a "local" scope:

- M memory: Any OB, FC, or FB can access the data in M memory, meaning that the data is available globally for all of the elements of the user program.
- Temp memory: Access to the data in temp memory is restricted to the OB, FC, or FB that created or declared the temp memory location. Temp memory locations remain local and are not shared by different code blocks, even when the code block calls another code block. For example: When an OB calls an FC, the FC cannot access the temp memory of the OB that called it.

The CPU provides temp (local) memory for each of the three OB priority groups:

- 16 Kbytes for startup and program cycle, including associated FBs and FCs
- 4 Kbytes for standard interrupt events including FBs and FCs
- 4 Kbytes for error interrupt events including FBs and FCs

You access temp memory by symbolic addressing only.

DB (data block): Use the DB memory for storing various types of data, including intermediate status of an operation or other control information parameters for FBs, and data structures required for many instructions such as timers and counters. You can access data block memory in bits, bytes, words, or double words. Both read and write access is permitted for read/write data blocks. Only read access is permitted for read-only data blocks.

Table 4- 14 Absolute addressing for DB memory

Bit	DB[data block number].DBX[byte address].[bit address]	DB1.DBX2.3
Byte, Word, or Double Word	DB[data block number].DB [size][starting byte address]	DB1.DBB4, DB10.DBW2, DB20.DBD8

Configuring the I/O in the CPU and I/O modules

Device overview

Module	Slot	I address	Q addre...	Type	Order
	103				
	102				
RS485_1	101			CM 1241 (RS485)	6ES7
PLC_1	1			CPU 1214C DC/DC/	6ES7
DI14/DO10	1.1	0...1	0...1	DI14/DO10	
AI2	1.2	64...67		AI2	
AO1 x 12bi.	1.3		80...81	AO1 signal board	6ES7
HSC_1	1.16	1000.....		High speed counte	
HSC_2	1.17			High speed counte	
HSC_3	1.18			High speed counte	
HSC_4	1.19			High speed counte	
HSC_5	1.20			High speed counte	
HSC_6	1.21			High speed counte	
Pulse_1	1.32			Pulse generator (P	
Pulse_2	1.33			Pulse generator (P	
PROFINET i...	X1			PROFINET interface	
DI8 x 24VDC...	2	8		SM 1221 DI8 x 24...	6ES7

When you add a CPU and I/O modules to your configuration screen, I and Q addresses are automatically assigned. You can change the default addressing by selecting the address field in the configuration screen and typing new numbers.

- Digital inputs and outputs are assigned in groups of 8 points (1 byte), whether the module uses all the points or not.
- Analog inputs and outputs are assigned in groups of 2 points (4 bytes).

The figure shows an example of a CPU 1214C with two SMs and one SB. In this example, you could change the address of the DI8 module to 2 instead of 8. The tool will assist you by changing address ranges that are the wrong size or conflict with other addresses.

4.3 Processing of analog values

Analog signal modules provide input signals or expect output values that represent either a voltage range or a current range. These ranges are ±10V, ±5V, ±2.5V, or 0 - 20mA. The values returned by the modules are integer values where 0 to 27648 represents the rated range for current, and -27648 to 27648 for voltage. Anything outside the range represents either an overflow or underflow. See the tables for analog input representation (Page 708) and analog output representation (Page 708) for details.

In your control program, you probably need to use these values in engineering units, for example to represent a volume, temperature, weight or other quantitative value. To do this for an analog input, you must first normalize the analog value to a real (floating point) value from 0.0 to 1.0. Then you must scale it to the minimum and maximum values of the engineering units that it represents. For values that are in engineering units that you need to convert to an analog output value, you first normalize the value in engineering units to a value between 0.0 and 1.0, and then scale it between 0 and 27648 or -27648 to 27648, depending on the range of the analog module. STEP 7 provides the NORM_X and SCALE_X instructions (Page 206) for this purpose. You can also use the CALCULATE instruction (Page 186) to scale the analog values (Page 30).

4.4 Data types

Data types are used to specify both the size of a data element as well as how the data are to be interpreted. Each instruction parameter supports at least one data type, and some parameters support multiple data types. Hold the cursor over the parameter field of an instruction to see which data types are supported for a given parameter.

A formal parameter is the identifier on an instruction that marks the location of data to be used by that instruction (example: the IN1 input of an ADD instruction). An actual parameter is the memory location or constant containing the data to be used by the instruction (example %MD400 "Number_of_Widgets"). The data type of the actual parameter specified by you must match one of the supported data types of the formal parameter specified by the instruction.

When specifying an actual parameter, you must specify either a tag (symbol) or an absolute (direct) memory address. Tags associate a symbolic name (tag name) with a data type, memory area, memory offset, and comment, and can be created either in the PLC tags editor or in the Interface editor for a block (OB, FC, FB and DB). If you enter an absolute address that has no associated tag, you must use an appropriate size that matches a supported data type, and a default tag will be created upon entry.

All data types except String are available in the PLC tags editor and the block Interface editors. String is available only in the block Interface editors. You can also enter a constant value for many of the input parameters.

- Bit and Bit sequences (Page 87): Bool (Boolean or bit value), Byte (8-bit byte value), Word (16-bit value), DWord (32-bit double-word value)
- Integer (Page 88)
  - USInt (unsigned 8-bit integer), SInt (signed 8-bit integer),
  - UInt (unsigned 16-bit integer), Int (signed 16-bit integer)
  - UDInt (unsigned 32-bit integer), DInt (signed 32-bit integer)
- Floating-point Real (Page 88): Real (32-bit Real or floating-point value), LReal (64-bit Real or floating-point value)
- Time and Date (Page 89): Time (32-bit IEC time value), Date (16-bit date value), TOD (32-bit time-off-day value), DT (64-bit date-and-time value)
- Character and String (Page 89): Char (8-bit single character), String (variable-length string of up to 254 characters)
- Array (Page 89)
- Data structure (Page 92): Struct
- PLC Data type (Page 93)
- Pointers (Page 94): Pointer, Any, Variant

Although not available as data types, the following BCD numeric format is supported by the conversion instructions.

Table 4- 15 Size and range of the BCD format

Format	Size (bits)	Numeric Range	Constant Entry Examples
BCD16	16	-999 to 999	123, -123
BCD32	32	-9999999 to 9999999	1234567, -1234567

4.4.1 Bool, Byte, Word, and DWord data types

Table 4- 16 Bit and bit sequence data types

Data type	Bit size	Number type	Number range	Constant examples	Address examples
Bool	1	Boolean	FALSE or TRUE	TRUE, 1,	I1.0 Q0.1 M50.7 DB1.DBX2.3 Tag_name
		Binary	0 or 1	0, 2#0	
		Octal	8#0 or 8#1	8#1	
		Hexadecimal	16#0 or 16#1	16#1	
Byte	8	Binary	2#0 to 2#11111111	2#00001111	IB2 MB10 DB1.DBB4 Tag_name
		Unsigned integer	0 to 255	15	
		Octal	8#0 to 8#377	8#17	
		Hexadecimal	B#16#0 to B#16#FF	B#16#F, 16#F	
Word	16	Binary	2#0 to 2#1111111111111111	2#1111000011110000	MW10 DB1.DBW2 Tag_name
		Unsigned integer	0 to 65535	61680	
		Octal	8#0 to 8#177777	8#170360	
		Hexadecimal	W#16#0 to W#16#FFFF, 16#0 to 16#FFFF	W#16#F0F0, 16#F0F0	
DWord	32	Binary	2#0 to 2#11111111111111111111111111111111	2#111100001111111100001111	MD10 DB1.DBD8 Tag_name
		Unsigned integer	0 to 4294967295	15793935	
		Octal	8#0 to 8#37777777777	8#74177417	
		Hexadecimal	DW#16#0000_0000 to DW#16#FFFF_FFFF, 16#0000_0000 to 16#FFFF_FFFF	DW#16#F0FF0F, 16#F0FF0F	

4.4.2 Integer data types

Table 4- 17 Integer data types (U = unsigned, S = short, D= double)

Data type	Bit size	Number Range	Constant examples	Address examples
USInt	8	0 to 255	78, 2#01001110	MB0, DB1.DBB4, Tag_name
SInt	8	-128 to 127	+50, 16#50	
UInt	16	0 to 65,535	65295, 0	MW2, DB1.DBW2, Tag_name
Int	16	-32,768 to 32,767	30000, +30000	
UDInt	32	0 to 4,294,967,295	4042322160	MD6, DB1.DBD8, Tag_name
DInt	32	-2,147,483,648 to 2,147,483,647	-2131754992	

4.4.3 Floating-point real data types

Real (or floating-point) numbers are represented as 32-bit single-precision numbers (Real), or 64-bit double-precision numbers (LReal) as described in the ANSI/IEEE 754-1985 standard. Single-precision floating-point numbers are accurate up to 6 significant digits and double-precision floating point numbers are accurate up to 15 significant digits. You can specify a maximum of 6 significant digits (Real) or 15 (LReal) when entering a floating-point constant to maintain precision.

Table 4- 18 Floating-point real data types (L=Long)

Data type	Bit size	Number range	Constant Examples	Address examples
Real	32	-3.402823e+38 to -1.175 495e-38, ±0, +1.175 495e-38 to +3.402823e+38	123.456, -3.4, 1.0e-5	MD100, DB1.DBD8, Tag_name
LReal	64	-1.7976931348623158e+308 to -2.2250738585072014e-308, ±0, +2.2250738585072014e-308 to +1.7976931348623158e+308	12345.123456789e40, 1.2E+40	DB_name.var_name Rules: • No direct addressing support • Can be assigned in an OB, FB, or FC block interface table • Can be assigned in a global or instance data block , only when the data block is created as the "optimized" type (symbolic access only)

Calculations that involve a long series of values including very large and very small numbers can produce inaccurate results. This can occur if the numbers differ by 10 to the power of x, where x > 6 (Real), or 15 (LReal). For example (Real): 100 000 000 + 1 = 100 000 000.

4.4.4 Time and Date data types

Table 4- 19 Time and date data types

Data type	Size	Range	Constant Entry Examples
Time	32 bits	T#-24d_20h_31m_23s_648ms to T#24d_20h_31m_23s_647ms Stored as: -2,147,483,648 ms to +2,147,483,647 ms	T#5m_30s T#1d_2h_15m_30s_45ms TIME#10d20h30m20s630ms 500h10000ms 10d20h30m20s630ms
Date	16 bits	D#1990-1-1 to D#2168-12-31	D#2009-12-31 DATE#2009-12-31 2009-12-31
Time_of_Day	32 bits	TOD#0:0:0.0 to TOD#23:59:59.999	TOD#10:20:30.400 TIME_OF_DAY#10:20:30.400 23:10:1
DTL (Date and Time Long)	12 bytes	Min.: DTL#1970-01-01-00:00:00.0 Max.: DTL#2554-12-31-23:59:59.999 999 999	DTL#2008-12-16-20:30:20.250

Time

TIME data is stored as a signed double integer interpreted as milliseconds. The editor format can use information for day (d), hours (h), minutes (m), seconds (s) and milliseconds (ms).

It is not necessary to specify all units of time. For example T#5h10s and 500h are valid.

The combined value of all specified unit values cannot exceed the upper or lower limits in milliseconds for the Time data type (-2,147,483,648 ms to +2,147,483,647 ms).

Date

DATE data is stored as an unsigned integer value which is interpreted as the number of days added to the base date 01/01/1990, to obtain the specified date. The editor format must specify a year, month and day.

TOD

TOD (TIME_OF_DAY) data is stored as an unsigned integer which is interpreted as the number of milliseconds since midnight for the specified time of day (Midnight = 0 ms). The hour (24hr/day), minute, and second must be specified. The fractional second specification is optional.

DTL

DTL (Date and Time Long) data type uses a12 byte structure that saves information on date and time. You can define DTL data in either the Temp memory of a block or in a DB. A value for all components must be entered in the "Start value" column of the DB editor.

Table 4- 20 Size and range for DTL

Length (bytes)	Format	Value range	Example of value input
12	Clock and calendar Year-Month-Day:Hour:Minute: Second.Nanoseconds	Min.: DTL#1970-01-01-00:00:00.0 Max.: DTL#2554-12-31-23:59:59.999 999 999	DTL#2008-12-16-20:30:20.250

Each component of the DTL contains a different data type and range of values. The data type of a specified value must match the data type of the corresponding components.

Table 4- 21 Elements of the DTL structure

Byte	Component	Data type	Value range
0	Year	UINT	1970 to 2554
1			
2	Month	USINT	1 to 12
3	Day	USINT	1 to 31
4	Weekday [1]	USINT	1(Sunday) to 7(Saturday) [1]
5	Hour	USINT	0 to 23
6	Minute	USINT	0 to 59
7	Second	USINT	0 to 59
8	Nanoseconds	UDINT	0 to 999 999 999
9			
10			
11			

[1] The weekday is not considered in the value entry.

Table 4- 22 Character and String data types

Data type	Size	Range	Constant Entry Examples
Char	8 bits	ASCII character codes: 16#00 to 16#FF	'A', 't', '@'
String	n+ 2 bytes	n = (0 to 254 character bytes)	'ABC'

Char

Char data occupies one byte in memory and stores a single character coded in ASCII format. The editor syntax uses a single quote character before and after the ASCII character. Visible characters and control characters can be used. A table of valid control characters is shown in the description of the String data type.

String

The CPU supports the String data type for storing a sequence of single-byte characters. The String data type contains a total character count (number of characters in the string) and the current character count. The String type provides up to 256 bytes for storing the maximum total character count (1 byte), the current character count (1 byte), and up to 254 characters, with each character stored in 1 byte.

You can use literal strings (constants) for instruction parameters of type IN using single quotes. For example, 'ABC' is a three-character string that could be used as input for parameter IN of the S_CONV instruction. You can also create string variables by selecting data type "String" in the block interface editors for OB, FC, FB, and DB. You cannot create a string in the PLC tags editor.

You can specify the maximum string size in bytes by entering square brackets after the keyword "String" (once the data type "String" is selected from a data type drop-list). For example, "MyString[10]" would specify a 10-byte maximum size for MyString. If you do not include the square brackets with a maximum size, then 254 is assumed.

The following example defines a String with maximum character count of 10 and current character count of 3. This means the String currently contains 3 one-byte characters, but could be expanded to contain up to 10 one-byte characters.

Table 4- 23 Example of a String data type

Total Character Count	Current Character Count	Character 1	Character 2	Character 3	...	Character 10
10	3	'C' (16#43)	'A' (16#41)	'T' (16#54)	...	-
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	...	Byte 11

ASCII control characters can be used in Char and String data. The following table shows examples of control character syntax.

Table 4- 24 Valid ASCII control characters

Control characters	ASCII Hex value	Control function	Examples
$L or $l	0A	Line feed	'$LText', '$0AText'
$N or $n	0A and 0D	Line break The new line shows two characters in the string.	'$NText', '$0A$0DText'
$P or $p	0C	Form feed	'$PText', '$0CText'
$R or $r	0D	Carriage return (CR)	'$RText','$0DText'
$T or $t	09	Tab	'$TText', '$09Text'
$$	24	Dollar sign	'100$$', '100$24'
$'	27	Single quote	'$'Text$'','$27Text$27'

Arrays

You can create an array that contains multiple elements of the same data type. Arrays can be created in the block interface editors for OB, FC, FB, and DB. You cannot create an array in the PLC tags editor.

To create an array from the block interface editor, name the array and choose data type "Array [lo .. hi] of type", then edit "lo", "hi", and "type" as follows:

- lo - the starting (lowest) index for your array
- hi - the ending (highest) index for your array
- type - one of the data types, such as BOOL, SINT, UDINT

Table 4- 25 ARRAY data type rules

Data Type	Array syntax		
ARRAY	Name [index1_min..index1_max, index2_min..index2_max] of <data type> • All array elements must be the same data type. • The index can be negative, but the lower limit must be less than or equal to the upper limit. • Arrays can have one to six dimensions. • Multi-dimensional index min..max declarations are separated by comma characters. • Nested arrays, or arrays of arrays, are not allowed. • The memory size of an array = (size of one element * total number of elements in array)		
	Array index	**Valid index data types**	**Array index rules**
	Constant or variable	USInt, SInt, UInt, Int, UDInt, DInt	• Value limits: -32768 to +32767 • Valid: Mixed constants and variables • Valid: Constant expressions • Not valid: Variable expressions

Example: array declarations	ARRAY[1..20] of REAL	One dimension, 20 elements
	ARRAY[-5..5] of INT	One dimension, 11 elements
	ARRAY[1..2, 3..4] of CHAR	Two dimensions, 4 elements
Example: array addresses	ARRAY1[0]	ARRAY1 element 0
	ARRAY2[1,2]	ARRAY2 element [1,2]
	ARRAY3[i,j]	If i =3 and j=4, then ARRAY3 element [3, 4] is addressed

4.4.5 Data structure data type

You can use the data type "Struct" to define a structure of data consisting of other data types. The struct data type can be used to handle a group of related process data as a single data unit. A Struct data type is named and the internal data structure declared in the data block editor or a block interface editor.

Arrays and structures can also be assembled into a larger structure. A structure can be nested up to eight levels deep. For example, you can create a structure of structures that contain arrays.

A Struct variable begins at an even-byte address and uses the memory to the next word boundary.

4.4.6 PLC data type

The PLC data type editor lets you define data structures that you can use multiple times in your program. You create a PLC data type by opening the "PLC data types" branch of the project tree and double-clicking the "Add new data type" item. On the newly created PLC data type item, use two single-clicks to rename the default name and double-click to open the PLC data type editor.

You create a custom PLC data type structure using the same editing methods that are used in the data block editor. Add new rows for any data types that are necessary to create the data structure that you want.

If a new PLC data type is created, then the new PLC type name will appear in the data type selector drop drop-lists in the DB editor and code block interface editor.

Potential uses of PLC data types:

- PLC data types can be used directly as a data type in a code block interface or in data blocks.
- PLC data types can be used as a template for the creation of multiple global data blocks that use the same data structure.

For example, a PLC data type could be a recipe for mixing colors. You can then assign this PLC data type to multiple data blocks. Each data block can then have the variables adjusted to create a specific color.

4.4.7 Pointer data types

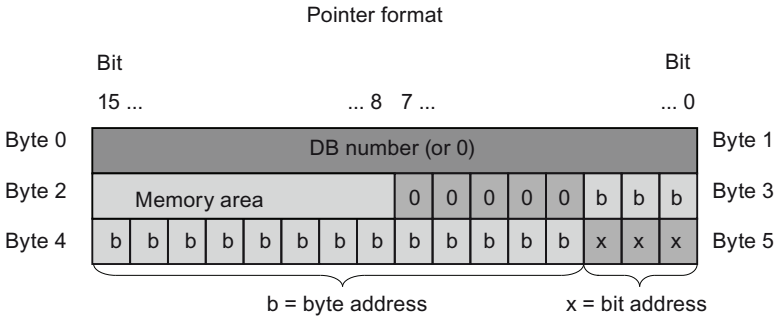
The pointer data types (Pointer, Any, and Variant) can be used in the block interface tables for FB and FC code blocks. You can select a pointer data type from the block interface data type drop-lists.

The Variant data type is also used for instruction parameters.

4.4.7.1 "Pointer" pointer data type

The data type Pointer points to a particular variable. It occupies 6 bytes (48 bits) in memory and can include the following information:

- DB number or 0 if the data is not stored in a DB
- Storage area in the CPU
- Variable address



Depending on the instruction, you can declare the following three types of pointers:

- Area-internal pointer: contains data on the address of a variable
- Area-crossing pointer: contains data on the memory area and the address of a variable
- DB-pointer: contains a data block number and the address of a variable

Table 4- 26 Pointer types:

Type	Format	Example entry
Area-internal pointer	P#Byte.Bit	P#20.0
Area-crossing pointer	P#Memory_area_Byte.Bit	P#M20.0
DB-pointer	P#Data_block.Data_element	P#DB10.DBX20.0

You can enter a parameter of type Pointer without the prefix (P #). Your entry will be automatically converted to the pointer format.

Table 4- 27 Memory area encoding in the Pointer data:

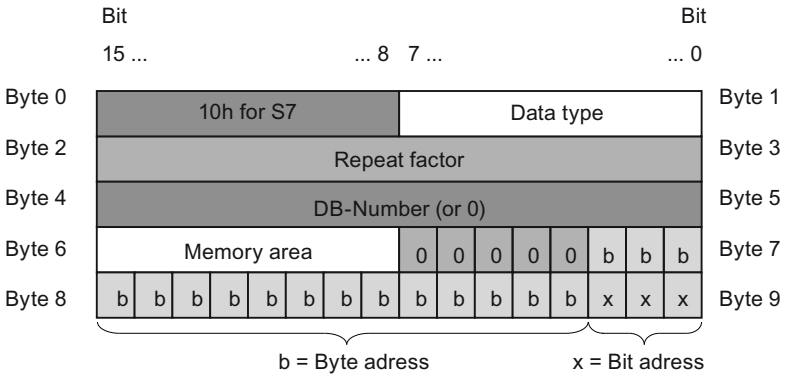
Hexadecimal code	Data type	Description
b#16#81	I	Input memory area
b#16#82	Q	Output memory area
b#16#83	M	Marker memory area
b#16#84	DBX	Data block
b#16#85	DIX	Instance data block
b#16#86	L	Local data
b#16#87	V	Previous local data

4.4.7.2 "Any" pointer data type

The pointer data type ANY ("Any") points to the beginning of a data area and specifies its length. The ANY pointer uses 10 bytes in memory and can include the following information:

- Data type: Data type of the data elements
- Repeat factor: Number of data elements
- DB Number: Data block in which data elements are stored
- Storage area: Memory area of the CPU, in which the data elements are stored
- Start address: "Byte.Bit" starting address of the data

The following image shows the structure of the ANY pointer:



A pointer can not detect ANY structures. It can only be assigned to local variables.

Table 4- 28 Format and examples of the ANY pointer:

Format	Entry example	Description
P#Data_block.Memory_area Data_address Type Number	P#DB 11.DBX 20.0 INT 10	10 words in global DB 11 starting from DBB 20.0
P#Memory_area Data_address Type Number	P#M 20.0 BYTE 10	10 bytes starting from MB 20.0
	P#I 1.0 BOOL 1	Input I1.0

Table 4- 29 Data type encoding in the ANY pointer

Hexadecimal code	Data type	Description
b#16#00	Null	Null pointer
b#16#01	Bool	Bits
b#16#02	Byte	Bytes, 8 Bits
b#16#03	Char	8-bit character
b#16#04	Word	16-bit-word
b#16#05	Int	16-bit-integer
b#16#37	SInt	8-bit-integer
b#16#35	UInt	16-bit unsigned integer
b#16#34	USInt	8-bit unsigned integer

Hexadecimal code	Data type	Description
b#16#06	DWord	32-bit double word
b#16#07	DInt	32-bit double integer
b#16#36	UDInt	32-bit-unsigned double integer
b#16#08	Real	32-Bit floating point
b#16#0B	Time	Time
b#16#13	String	Character string

Table 4- 30 Memory area encoding in the ANY pointer:

Hexadecimal code	Memory area	Description
b#16#81	I	Input memory area
b#16#82	Q	Output memory area
b#16#83	M	Marker memory area
b#16#84	DBX	Data block
b#16#85	DIX	Instance data block
b#16#86	L	Local data
b#16#87	V	Previous local data

4.4.7.3 "Variant" pointer data type

The data type Variant is can point to variables of different data types or parameters. The Variant pointer can point to structures and individual structural components. The Variant pointer does not occupy any space in memory.

Table 4- 31 Properties of the Variant pointer

Length (Byte)	Representation	Format	Example entry
0	Symbolic	Operand	MyTag
		DB_name.Struct_name.element_name	MyDB.Struct1.pressure1
	Absolute	Operand	%MW10
		DB_number.Operand Type Length	P#DB10.DBX10.0 INT 12

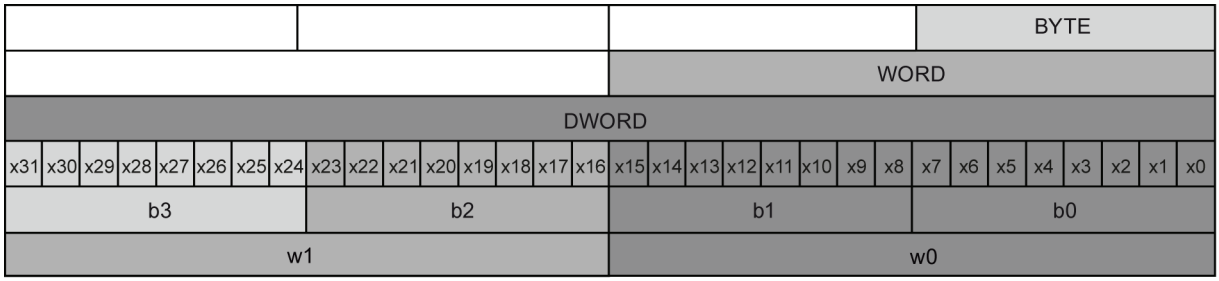
4.4.8 Accessing a "slice" of a tagged data type

PLC tags and data block tags can be accessed at the bit, byte, or word level depending on their size. The syntax for accessing such a data slice is as follows:

- "<PLC tag name>".xn (bit access)
- "<PLC tag name>".bn (byte access)
- "<PLC tag name>".wn (word access)
- "<Data block name>".<tag name>.xn (bit access)

- "<Data block name>".<tag name>.bn (byte access)
- "<Data block name>".<tag name>.wn (word access)

A double word-sized tag can be accessed by bits 0 - 31, bytes 0 - 3, or word 0 - 1. A word-sized tag can be accessed by bits 0 - 15, bytes 0 - 2, or word 0. A byte-sized tag can be accessed by bits 0 - 8, or byte 0. Bit, byte, and word slices can be used anywhere that bits, bytes, or words are expected operands.



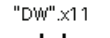
Note

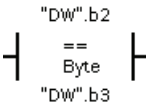
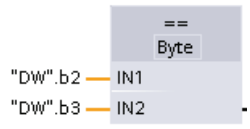
Valid data types that can be accessed by slice are Byte, Char, Conn_Any, Date, DInt, DWord, Event_Any, Event_Att, Hw_Any, Hw_Device, HW_Interface, Hw_Io, Hw_Pwm, Hw_SubModule, Int, OB_Any, OB_Att, OB_Cyclic, OB_Delay, OB_WHINT, OB_PCYCLE, OB_STARTUP, OB_TIMEERROR, OB_Tod, Port, Rtm, SInt, Time, Time_Of_Day, UDInt, UInt, USInt, and Word. PLC Tags of type Real can be accessed by slice, but data block tags of type Real cannot.

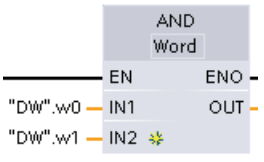
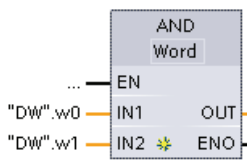
Examples

In the PLC tag table, "DW" is a declared tag of type DWORD. The examples show bit, byte, and word slice access:

	LAD	FBD	SCL

Bit access			`IF "DW".x11 THEN`
			`...`
			`END_IF;`

Byte access			`IF "DW".b2 = "DW".b3 THEN`
			`...`
			`END_IF;`

Word access			`out:= "DW".w0 AND "DW".w1;`

See SCL (Page 145) for syntax for addressing local variables and PLC tags.

4.4.9 Accessing a tag with an AT overlay

The AT tag overlay allows you to access an already-declared tag of a standard access block with an overlaid declaration of a different data type. You can, for example, address the individual bits of a tag of a Byte, Word, or DWord data type with an Array of Bool.

Declaration

To overlay a parameter, declare an additional parameter directly after the parameter that is to be overlaid and select the data type "AT". The editor creates the overlay, and you can then choose the data type, struct, or array that you wish to use for the overlay.

Example

This example shows the input parameters of a standard-access FB. The byte tag B1 is overlaid with an array of Booleans:

■	B1		Byte
▼	AT	AT "B1"	Array [0..7] of Bool
■	AT[0]		Bool
■	AT[1]		Bool
■	AT[2]		Bool
■	AT[3]		Bool
■	AT[4]		Bool
■	AT[5]		Bool
■	AT[6]		Bool
■	AT[7]		Bool

Table 4- 32 Overlay of a byte with a Boolean array

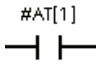
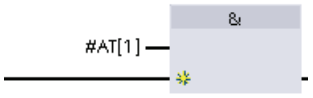
7	6	5	4	3	2	1	0
AT[0]	AT[1]	AT[2]	AT[3]	AT[4]	AT[5]	AT[6]	AT[7]

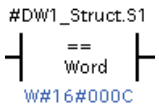
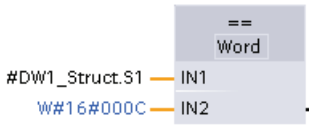
Another example is a DWord tag overlaid with a Struct:

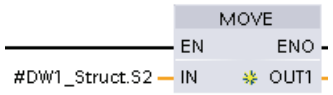
■	DW1		DWord
▼	DW1_Struct	AT "DW1"	Struct
■	S1		Word
■	S2		Byte
■	S3		Byte

The overlay types can be addressed directly in the program logic:

LAD	FBD	SCL

		`IF #AT[1] THEN`
		`...`
		`END_IF;`

		`IF (#DW1_Struct.S1 =`
		`W#16#000C) THEN`
		`...`
		`END_IF;`

		`out1 := #DW1_Struct.S2;`

See SCL (Page 145) for syntax for addressing local variables and PLC tags.

Rules

- Overlaying of tags is only possible in FB and FC blocks with standard access.
- You can overlay parameters for all block types and all declaration sections.
- An overlaid parameter can be used like any other block parameter.
- You cannot overlay parameters of type VARIANT.
- The size of the overlaying parameter must be less than or equal to the size of the overlaid parameter.
- The overlaying variable must be declared immediately after the variable that it overlays and identified with the keyword "AT".

4.5 Using a memory card

NOTICE
The CPU supports only the pre-formatted SIMATIC memory card (Page 758). Before you copy any program to the formatted memory card, delete any previously saved program from the memory card.

Use the memory card either as a transfer card or as a program card. Any program that you copy to the memory card contains all of the code blocks and data blocks, any technology objects, and the device configuration. The program does **not** contain force values.

- Use a transfer card to copy a program to the internal load memory of the CPU without using STEP 7. After you insert the transfer card, the CPU first erases the user program and any force values from the internal load memory, and then copies the program from the transfer card to the internal load memory. When the transfer process is complete, you must remove the transfer card.

  You can use an empty transfer card to access a password-protected CPU when the password has been lost or forgotten (Page 107). Inserting the empty transfer card deletes the password-protected program in the internal load memory of the CPU. You can then download a new program to the CPU.

- Use a program card as external load memory for the CPU. Inserting a program card in the CPU erases all of the CPU internal load memory (the user program and any force values). The CPU then executes the program in external load memory (the program card). Downloading to a CPU that has a program card updates only the external load memory (the program card).

  Because the internal load memory of the CPU was erased when you inserted the program card, the program card **must** remain in the CPU. If you remove the program card, the CPU goes to STOP mode. (The error LED flashes to indicate that program card has been removed.)

The program on a memory card includes the code blocks, the data blocks, the technology objects, and the device configuration. The memory card does **not** contain any force values. The force values are not part of the program, but are stored in the load memory, whether the internal load memory of the CPU, or the external load memory (a program card). If a program card is inserted in the CPU, then STEP 7 applies the force values only to the external load memory on the program card.

You also use a memory card when downloading firmware updates from customer support (http://www.siemens.com/automation/support-request). A firmware update requires a 24 MB memory card.

4.5.1 Inserting a memory card in the CPU

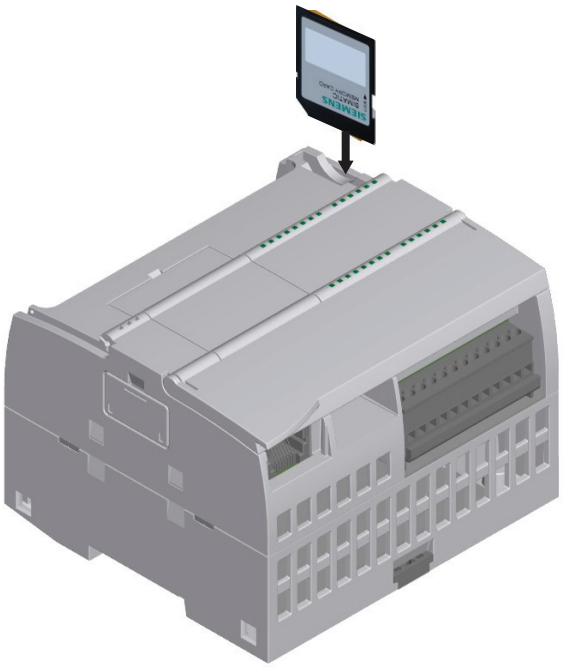
CAUTION
Electrostatic discharge can damage the memory card or the receptacle on the CPU. Make contact with a grounded conductive pad and/or wear a grounded wrist strap when you handle the memory card. Store the memory card in a conductive container.

Check that the memory card is not write-protected. Slide the protection switch away from the "Lock" position.

CAUTION
If you insert a memory card (whether configured as a program or transfer card) into a running CPU, the CPU goes immediately to STOP mode, which might result damage to the equipment or to the process being controlled. Before inserting or removing a memory card, always ensure that the CPU is not actively controlling a machine or process. Always install an emergency stop circuit for your application or process.

Note

If you insert a memory card with the CPU in STOP mode, the diagnostic buffer displays a message that the memory card evaluation has been initiated. The CPU will evaluate the memory card the next time you either change the CPU to RUN mode, reset the CPU memory with an MRES, or power-cycle the CPU.

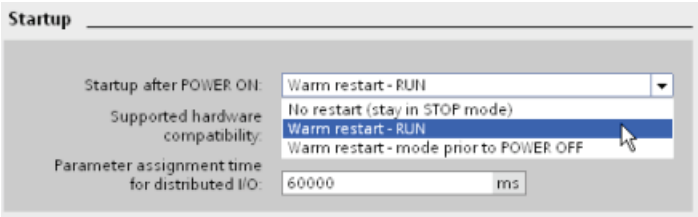


To insert a memory card, open the top CPU door and insert the memory card in the slot. A push-push type connector allows for easy insertion and removal.

The memory card is keyed for proper installation.

4.5.2 Configuring the startup parameter of the CPU before copying the project to the memory card

When you copy a program to a transfer card or a program card, the program includes the startup parameter for the CPU. Before copying the program to the memory card, always ensure that you have configured the operating mode for the CPU following a power-cycle. Select whether the CPU starts in STOP mode, RUN mode, or in the previous mode (prior to the power cycle).



4.5.3 Transfer card

CAUTION
Electrostatic discharge can damage the memory card or the receptacle on the CPU. Make contact with a grounded conductive pad and/or wear a grounded wrist strap whenever you handle the memory card. Store the memory card in a conductive container.

Creating a transfer card

Always remember to configure the startup parameter of the CPU (Page 102) before copying a program to the transfer card. To create a transfer card, follow these steps:

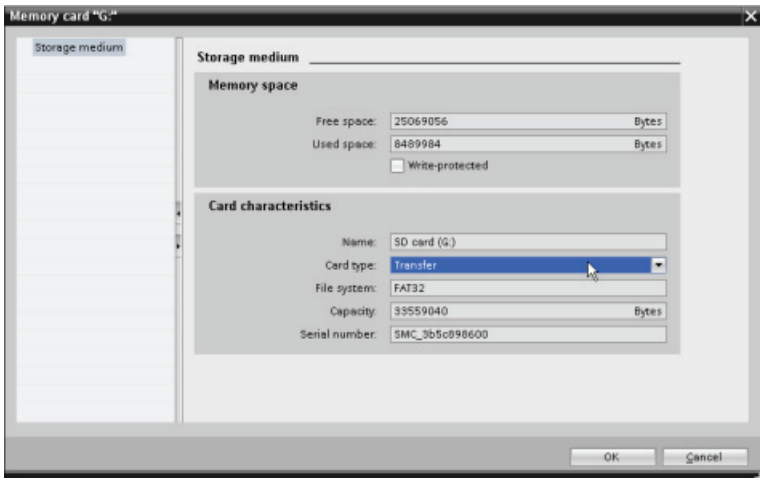
1. Insert a blank SIMATIC memory card into an SD card reader/writer attached to your computer.

   If you are reusing a SIMATIC memory card that contains a user program or a firmware update, you **must** delete the program files before reusing the card. Use Windows Explorer to display the contents of the memory card and delete the "S7_JOB.S7S" file and also delete any directory folder (such as "SIMATIC.S7S" or "FWUPDATE.S7S").

2. In the Project tree (Project view), expand the "SIMATIC Card Reader" folder and select your card reader.
3. Display the "Memory card" dialog by right-clicking the drive letter corresponding to the memory card in the card reader and selecting "Properties" from the context menu.

4. In the "Memory card" dialog, select "Transfer" from the "Card type" drop-down menu.

   At this point, STEP 7 creates the empty transfer card. If you are creating an empty transfer card, such as to recover from a lost CPU password (Page 107), remove the transfer card from the card reader.



5. Add the program by selecting the CPU device (such as PLC_1 [CPU 1214 DC/DC/DC]) in the Project tree and dragging the CPU device to the memory card. (Another method is to copy the CPU device and paste it to the memory card.) Copying the CPU device to the memory card opens the "Load preview" dialog.
6. In the "Load preview" dialog, click the "Load" button to copy the CPU device to the memory card.
7. When the dialog displays a message that the CPU device (program) has been loaded without errors, click the "Finish" button.

Using a transfer card

WARNING

Verify that the CPU is not actively running a process before inserting the memory card.

Inserting a memory card will cause the CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Before inserting a memory card, always ensure that the CPU is offline and in a safe state.

To transfer the program to a CPU, follow these steps:

1. Insert the transfer card into the CPU (Page 100). If the CPU is in RUN, the CPU will go to STOP mode. The maintenance (MAINT) LED flashes to indicate that the memory card needs to be evaluated.
2. Power-cycle the CPU to evaluate the memory card. Alternative methods for rebooting the CPU are to perform either a STOP-to-RUN transition or a memory reset (MRES) from STEP 7.
3. After the rebooting and evaluating the memory card, the CPU copies the program to the internal load memory of the CPU.

   The RUN/STOP LED alternately flashes green and yellow to indicate that the program is being copied. When the RUN/STOP LED turns on (solid yellow) and the MAINT LED flashes, the copy process has finished. You can then remove the memory card.
4. Reboot the CPU (either by restoring power or by the alternative methods for rebooting) to evaluate the new program that was transferred to internal load memory.

The CPU then goes to the start-up mode (RUN or STOP) that you configured for the project.

Note

You must remove the transfer card before setting the CPU to RUN mode.

4.5.4 Program card

CAUTION
Electrostatic discharge can damage the memory card or the receptacle on the CPU. Make contact with a grounded conductive pad and/or wear a grounded wrist strap when you handle the memory card. Store the memory card in a conductive container.

Check that the memory card is not write-protected. Slide the protection switch away from the "Lock" position.

Before you copy any program elements to the program card, delete any previously saved programs from the memory card.

Creating a program card

When used as a program card, the memory card is the external load memory of the CPU. If you remove the program card, the internal load memory of the CPU is empty.

Note

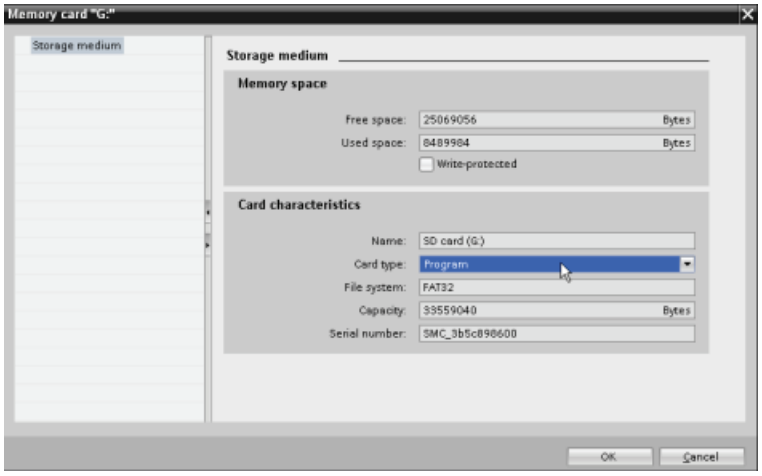
If you insert a blank memory card into the CPU and perform a memory card evaluation by either power cycling the CPU, performing a STOP to RUN transition, or performing a memory reset (MRES), the program and force values in internal load memory of the CPU are copied to the memory card. (The memory card is now a program card.) After the copy has been completed, the program in internal load memory of the CPU is then erased. The CPU then goes to the configured startup mode (RUN or STOP).

Always remember to configure the startup parameter of the CPU (Page 102) before copying a project to the program card. To create a program card, follow these steps:

1. Insert a blank SIMATIC memory card into an SD card reader/writer attached to your computer.

   If you are reusing a SIMATIC memory card that contains a user program or a firmware update, you **must** delete the program files before reusing the card. Use Windows Explorer to display the contents of the memory card and delete the "S7_JOB.S7S" file and also delete any directory folder (such as "SIMATIC.S7S" or "FWUPDATE.S7S").

2. In the Project tree (Project view), expand the "SIMATIC Card Reader" folder and select your card reader.
3. Display the "Memory card" dialog by right-clicking the drive letter corresponding to the memory card in the card reader and selecting "Properties" from the context menu.
4. In the "Memory card" dialog, select "Program" from the drop-down menu.



5. Add the program by selecting the CPU device (such as PLC_1 [CPU 1214 DC/DC/DC]) in the Project tree and dragging the CPU device to the memory card. (Another method is to copy the CPU device and paste it to the memory card.) Copying the CPU device to the memory card opens the "Load preview" dialog.

6. In the "Load preview" dialog, click the "Load" button to copy the CPU device to the memory card.
7. When the dialog displays a message that the CPU device (program) has been loaded without errors, click the "Finish" button.

Using a program card as the load memory for your CPU

WARNING

Verify that the CPU is not actively running a process before inserting the memory card.

Inserting a memory card will cause the CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Before inserting a memory card, always ensure that the CPU is offline and in a safe state.

To use a program card with your CPU, follow these steps:

1. Insert the program card into the CPU. If the CPU is in RUN mode, the CPU goes to STOP mode. The maintenance (MAINT) LED flashes to indicate that the memory card needs to be evaluated.
2. Power-cycle the CPU to evaluate the memory card. Alternative methods for rebooting the CPU are to perform either a STOP-to-RUN transition or a memory reset (MRES) from STEP 7.
3. After the CPU reboots and evaluates the program card, the CPU erases the internal load memory of the CPU.

The CPU then goes to the start-up mode (RUN or STOP) that you configured for the CPU.

The program card must remain in the CPU. Removing the program card leaves the CPU with no program in internal load memory.

WARNING

If you remove the program card, the CPU loses its external load memory and generates an error. The CPU goes to STOP mode and flashes the error LED.

Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment.

4.6 Recovery from a lost password

If you have lost the password for a password-protected CPU, use an empty transfer card to delete the password-protected program. The empty transfer card erases the internal load memory of the CPU. You can then download a new user program from STEP 7 to the CPU.

For information about the creation and use of an empty transfer card, see the section of transfer cards (Page 102).

⚠ WARNING

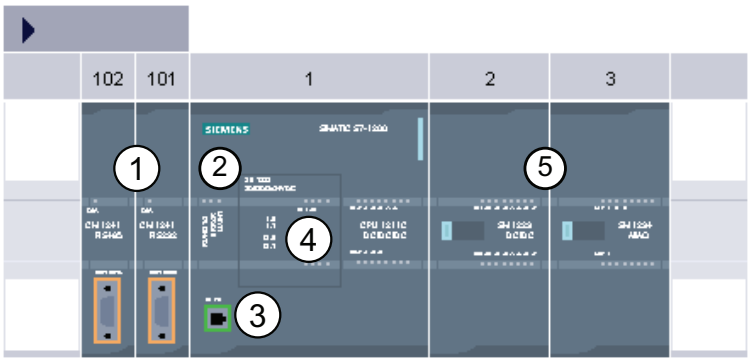
If you insert a transfer card in a running CPU, the CPU goes to STOP. Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment.

You must remove the transfer card before setting the CPU to RUN mode.

Device configuration

5

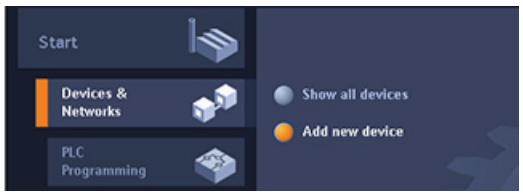
You create the device configuration for your PLC by adding a CPU and additional modules to your project.



① Communication module (CM) or communication processor (CP): Up to 3, inserted in slots 101, 102, and 103
② CPU: Slot 1
③ Ethernet port of CPU
④ Signal board (SB) or communication board (CB): up to 1, inserted in the CPU
⑤ Signal module (SM) for digital or analog I/O: up to 8, inserted in slots 2 through 9 (CPU 1214C allows 8, CPU 1212C allows 2, CPU 1211C does not allow any)

To create the device configuration, add a device to your project.

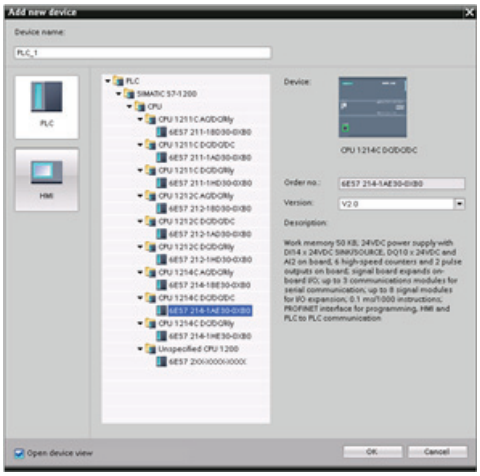
- In the Portal view, select "Devices & Networks" and click "Add new device".
- In the Project view, under the project name, double-click "Add new device".



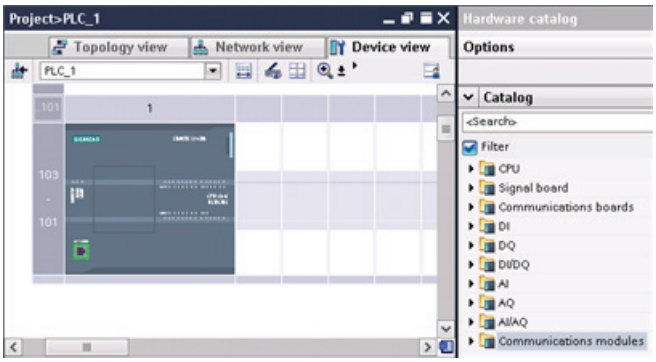
5.1 Inserting a CPU

You create your device configuration by inserting a CPU into your project. Selecting the CPU from the "Add new device" dialog creates the rack and CPU.

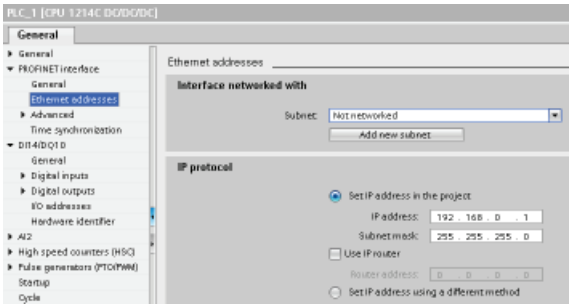
"Add new device" dialog



Device view of the hardware configuration



Selecting the CPU in the Device view displays the CPU properties in the inspector window.



Note

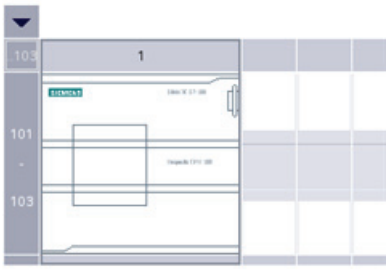
The CPU does not have a pre-configured IP address. You must manually assign an IP address for the CPU during the device configuration. If your CPU is connected to a router on the network, you also enter the IP address for a router.

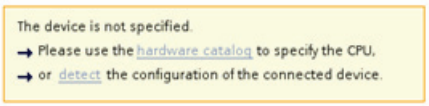
5.2 Detecting the configuration for an unspecified CPU

If you are connected to a CPU, you can upload the configuration of that CPU, including any modules, to your project. Simply create a new project and select the "unspecified CPU" instead of selecting a specific CPU. (You can also skip the device configuration entirely by selecting the "Create a PLC program" from the "First steps". STEP 7 then automatically creates an unspecified CPU.)

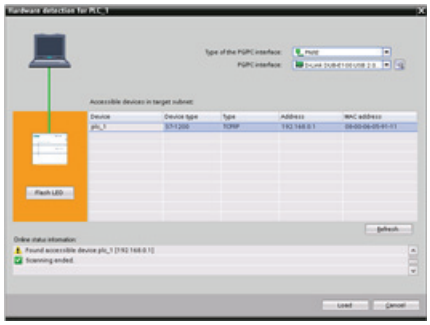
From the program editor, you select the "Hardware detection" command from the "Online" menu.

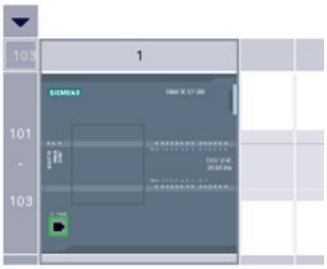
From the device configuration editor, you select the option for detecting the configuration of the connected device.





After you select the CPU from the online dialog and click the Load button, STEP 7 uploads the hardware configuration from the CPU, including any modules (SM, SB, or CM). You can then configure the parameters for the CPU and the modules.





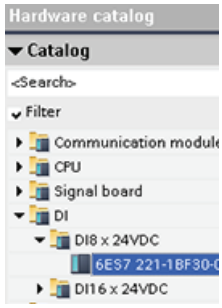
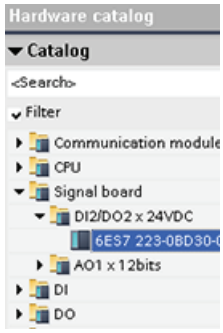
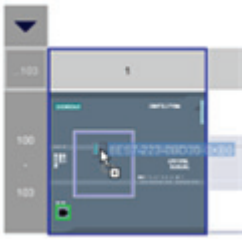
5.3 Adding modules to the configuration

Use the hardware catalog to add modules to the CPU:

- Signal module (SM) provides additional digital or analog I/O points. These modules are connected to the right side of the CPU.
- Signal board (SB) provides just a few additional I/O points for the CPU. The SB is installed on the front of the CPU.
- Communication board (CB) provides an additional communication port (such as RS485). The CB is installed on the front of the CPU.
- Communication module (CM) and communication processor (CP) provide an additional communication port, such as for PROFIBUS or GPRS. These modules are connected to the left side of the CPU.

To insert a module into the hardware configuration, select the module in the hardware catalog and either double-click or drag the module to the highlighted slot.

Table 5- 1 Adding a module to the device configuration

Module	Select the module	Insert the module	Result
SM			
SB or CB			
CM or CP			

5.4 Configuring the operation of the CPU

To configure the operational parameters for the CPU, select the CPU in the Device view (blue outline around whole CPU), and use the "Properties" tab of the inspector window.

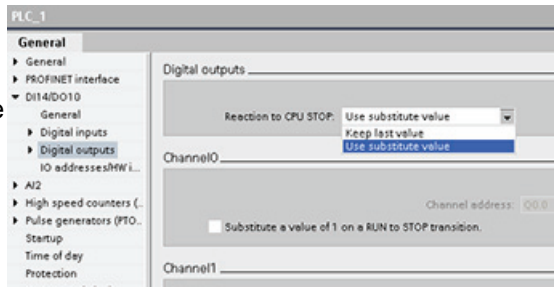


Table 5- 2 CPU properties

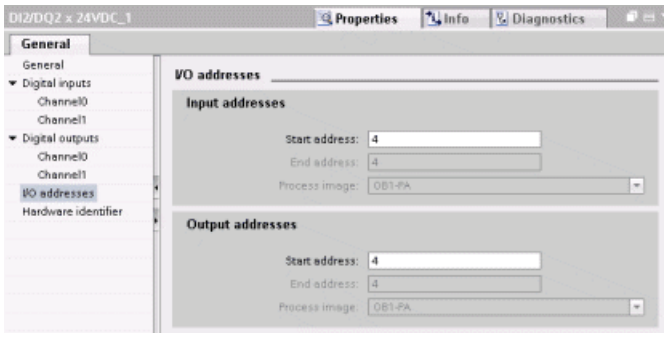
Property	Description
PROFINET interface	Sets the IP address for the CPU and time synchronization
DI, DO, and AI	Configures the behavior of the local (on-board) digital and analog I/O
High-speed counters (Page 315) and pulse generators (Page 289)	Enables and configures the high-speed counters (HSC) and the pulse generators used for pulse-train operations (PTO) and pulse-width modulation (PWM) When you configure the outputs of the CPU or signal board as pulse generators (for use with the PWM or basic motion control instructions), the corresponding output addresses (Q0.0, Q0.1, Q4.0, and Q4.1) are removed from the Q memory and cannot be used for other purposes in your user program. If your user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.
Startup (Page 63)	**Startup after POWER ON:** Selects the behavior of the CPU following an off-to-on transition, such as to start in STOP mode or to go to RUN mode after a warm restart
	Supported hardware compatibility: Configures whether modules (SM, SB, CM, CP or even the CPU) can be substituted: • Allow acceptable substitute (default) • Allow any substitute For example, a 16-channel SM could be an acceptable substitute for an 8-channel I/O.
	Parameter assignment time for distributed I/O: Configures a maximum amount of time (default: 60000 ms) for the distributed I/O to be brought online. (The CMs and CPs receive power and communication parameters from the CPU during startup. This assignment time allows time for the I/O connected to the CM or CP be brought online.) The CPU goes to RUN as soon as the distributed I/O is online, regardless of the assignment time. If the distributed I/O has not been brought online within this time, the CPU still goes to RUN--without the distributed I/O. **Note:** If your configuration uses a CM 1243-5 (PROFIBUS master), do not set this parameter below 15 seconds (15000 ms) to ensure that the module to be brought online.
Cycle (Page 73)	Defines a maximum cycle time or a fixed minimum cycle time
Communication load	Allocates a percentage of the CPU time to be dedicated to communication tasks
System and clock memory (Page 78)	Enables a byte for "system memory" functions and enables a byte for "clock memory" functions (where each bit toggles on and off at a predefined frequency).
Web server (Page 473)	Enables and configures the Web server feature.
Time of day	Selects the time zone and configures daylight saving time

Property	Description
Protection (Page 153)	Sets the read/write protection and password for accessing the CPU
Connection resources (Page 394)	Provides a summary of the communication connections that are available for the CPU and the number of connections that have been configured.
Overview of addresses	Provides a summary of the I/O addresses that have been configured for the CPU.

5.5 Configuring the parameters of the modules

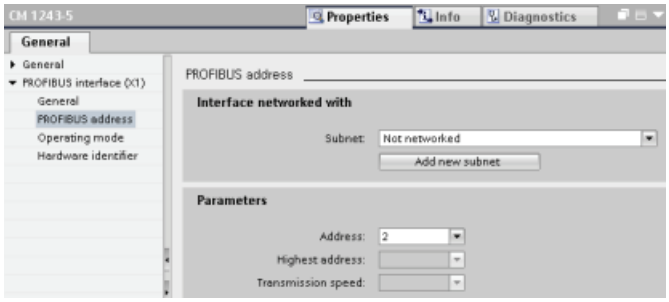
To configure the operational parameters for the modules, select the module in the Device view and use the "Properties" tab of the inspector window to configure the parameters for the module.

Configuring a signal module (SM) or a signal board (SB)



- Digital I/O: Inputs can be configured for rising-edge detection or falling-edge detection (associating each with an event and hardware interrupt) and also for "pulse catch" (to stay on after a momentary pulse) through the next update of the input process image. Outputs can use a freeze or substitute value.
- Analog I/O: For individual inputs, configure parameters, such as measurement type (voltage or current), range and smoothing, and to enable underflow or overflow diagnostics. Analog outputs provide parameters such as output type (voltage or current) and for diagnostics, such as short-circuit (for voltage outputs) or upper/lower limit diagnostics. You do not configure ranges of analog inputs and outputs in engineering units on the Properties dialog. You must handle this in your program logic as described in the topic "Processing of analog values (Page 85)".
- I/O diagnostic addresses: Configures the start address for the set of inputs and outputs of the module

Configuring a communication interface (CM, CP or CB)



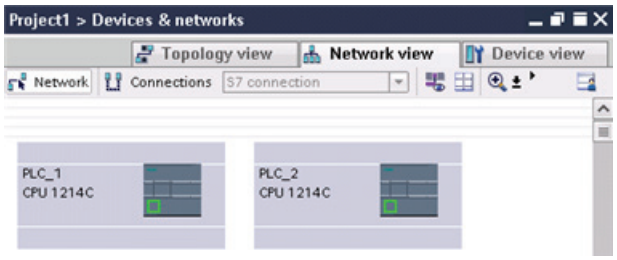
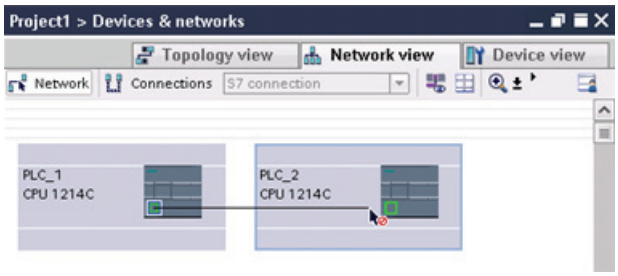
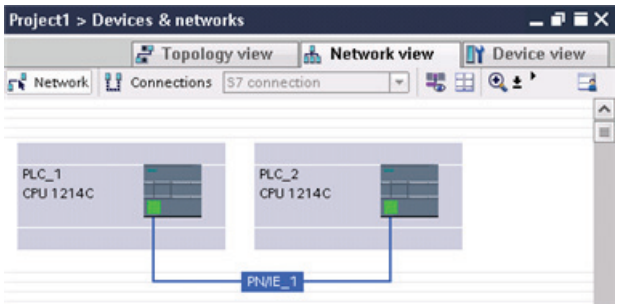
Depending on the type of communication interface, you configure the parameters for the network.

5.6 Configuring the CPU for communication

5.6.1 Creating a network connection

Use the "Network view" of Device configuration to create the network connections between the devices in your project. After creating the network connection, use the "Properties" tab of the inspector window to configure the parameters of the network.

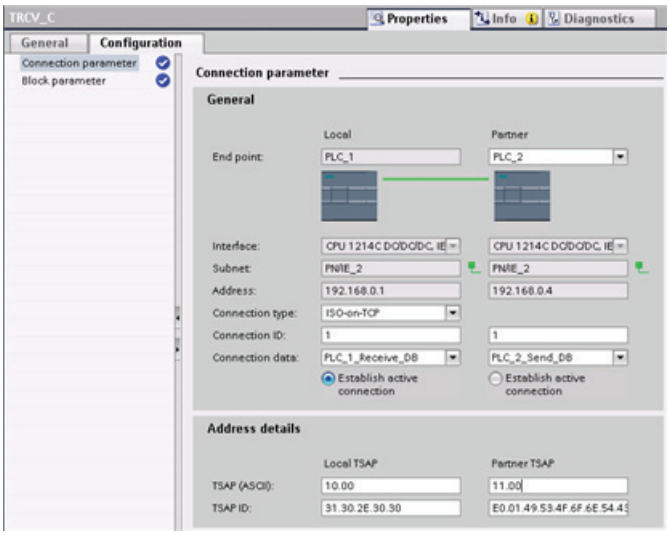
Table 5- 3 Creating a network connection

Action	Result
Select "Network view" to display the devices to be connected.	
Select the port on one device and drag the connection to the port on the second device.	
Release the mouse button to create the network connection.	

5.6.2 Configuring the Local/Partner connection path

The inspector window displays the properties of the connection whenever you have selected any part of the instruction. Specify the communication parameters in the "Configuration" tab of the "Properties" for the communication instruction.

Table 5- 4 Configuring the connection path (using the properties of the instruction)

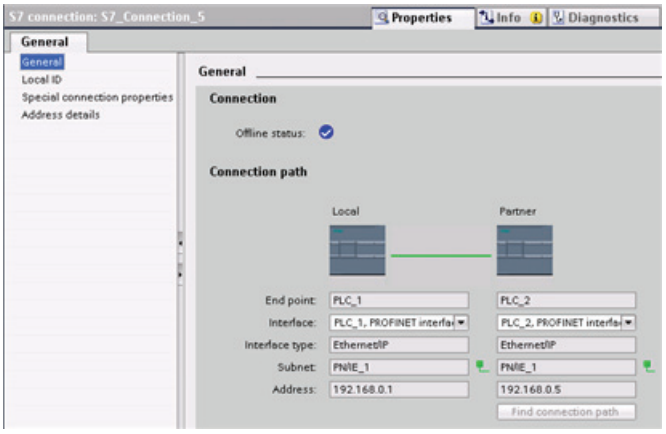
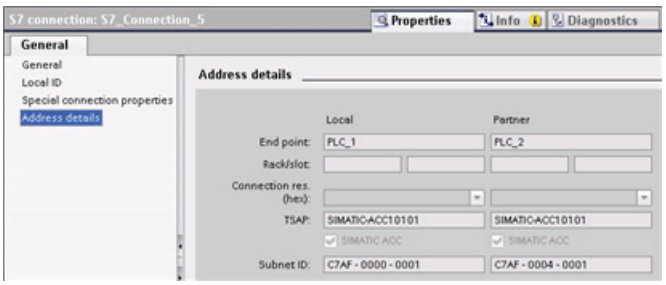
TCP, ISO-on-TCP, and UDP	Connection properties
For the TCP, ISO-on-TCP, and UDP Ethernet protocols, use the "Properties" of the instruction (TSEND_C, TRCV_C, or TCON) to configure the "Local/Partner" connections. The illustration shows the "Connection properties" of the "Configuration tab" for an ISO-on-TCP connection.	

Note

When you configure the connection properties for one CPU, STEP 7 allows you either to select a specific connection DB in the partner CPU (if one exists), or to create the connection DB for the partner CPU. The partner CPU must already have been created for the project and cannot be an "unspecified" CPU.

You must still insert a TSEND_C, TRCV_C or TCON instruction into the user program of the partner CPU. When you insert the instruction, select the connection DB that was created by the configuration.

Table 5- 5 Configuring the connection path for S7 communication (Device configuration)

S7 communication (GET and PUT)	Connection properties
For S7 communication, use the "Devices & networks" editor of the network to configure the Local/Partner connections. You can click the "Highlighted: Connection" button to access the "Properties". The "General" tab provides several properties: • "General" (shown) • "Local ID" • "Special connection properties" • "Address details" (shown)	 

Refer to "Protocols" (Page 399) in the "PROFINET" section or to "Creating an S7 connection" (Page 464) in the "S7 communication" section for more information and a list of available communication instructions.

Table 5- 6 Parameters for the multiple CPU connection

Parameter		Definition
Address		Assigned IP addresses
General	End point	Name assigned to the partner (receiving) CPU
	Interface	Name assigned to the interfaces
	Subnet	Name assigned to the subnets
	Interface type	*S7 communication only*: Type of interface
	Connection type	Type of Ethernet protocol
	Connection ID	ID number
	Connection data	Local and Partner CPU data storage location
	Establish active connection	Radio button to select Local or Partner CPU as the active connection
Address details	End point	*S7 communication only*: Name assigned to the partner (receiving) CPU
	Rack/slot	*S7 communication only*: Rack and slot location
	Connection resource	*S7 communication only*: Component of the TSAP used when configuring an S7 connection with an S7-300 or S7-400 CPU
	Port (decimal):	TCP and UPD: Partner CPU port in decimal format

Parameter		Definition
	TSAP [1] and Subnet ID:	ISO on TCP (RFC 1006) and S7 communication: Local and partner CPU TSAPs in ASCII and hexadecimal formats

[1] When configuring a connection with an S7-1200 CPU for ISO-on-TCP, use only ASCII characters in the TSAP extension for the passive communication partners.

Transport Service Access Points (TSAPs)

Using TSAPs, ISO on TCP protocol and S7 communication allows multiple connections to a single IP address (up to 64K connections). TSAPs uniquely identify these communication end point connections to an IP address.

In the "Address Details" section of the Connection Parameters dialog, you define the TSAPs to be used. The TSAP of a connection in the CPU is entered in the "Local TSAP" field. The TSAP assigned for the connection in your partner CPU is entered under the "Partner TSAP" field.

Port Numbers

With TCP and UDP protocols, the connection parameter configuration of the Local (active) connection CPU must specify the remote IP address and port number of the Partner (passive) connection CPU.

In the "Address Details" section of the Connection Parameters dialog, you define the ports to be used. The port of a connection in the CPU is entered in the "Local Port" field. The port assigned for the connection in your partner CPU is entered under the "Partner Port" field.

5.6.3 Parameters for the PROFINET connection

The TSEND_C, TRCV_C and TCON instructions require that connection-related parameters be specified in order to connect to the partner device. These parameters are specified by the TCON_Param structure for the TCP, ISO-on-TCP and UDP protocols. Typically, you use the "Configuration" tab of the "Properties" (Page 118) of the instruction to specify these parameters. If the "Configuration" tab is not accessible, then you must specify the TCON_Param structure programmatically.

Table 5- 7 Structure of the connection description (TCON_Param)

Byte	Parameter and data type		Description
0 … 1	block_length	UInt	Length: 64 bytes (fixed)
2 … 3	id	CONN_OUC (Word)	Reference to this connection: Range of values: 1 (default) to 4095. Specify the value of this parameter for the TSEND_C, TRCV_C or TCON instruction under ID.
4	connection_type	USInt	Connection type: • 17: TCP (default) • 18: ISO-on-TCP • 19: UDP

Byte	Parameter and data type		Description
5	active_est	Bool	ID for the type of connection: • TCP and ISO-on-TCP: – FALSE: Passive connection – TRUE: Active connection (default) • UDP: FALSE
6	local_device_id	USInt	ID for the local PROFINET or Industrial Ethernet interface: 1 (default)
7	local_tsap_id_len	USIn	Length of parameter local_tsap_id used, in bytes; possible values: • TCP: 0 (active, default) or 2 (passive) • ISO-on-TCP: 2 to 16 • UDP: 2
8	rem_subnet_id_len	USInt	This parameter is not used.
9	rem_staddr_len	USInt	Length of address of partner end point, in bytes: • 0: unspecified (parameter rem_staddr is irrelevant) • 4 (default): Valid IP address in parameter rem_staddr (only for TCP and ISO-on-TCP)
10	rem_tsap_id_len	USInt	Length of parameter rem_tsap_id used, in bytes; possible values: • TCP: 0 (passive) or 2 (active, default) • ISO-on-TCP: 2 to 16 • UDP: 0
11	next_staddr_len	USInt	This parameter is not used.
12 … 27	local_tsap_id	Array [1..16] of Byte	Local address component of connection: • TCP and ISO-on-TCP: local port no. (possible values: 1 to 49151; recommended values: 2000...5000): – local_tsap_id[1] = high byte of port number in hexadecimal notation; – local_tsap_id[2] = low byte of port number in hexadecimal notation; – local_tsap_id[3-16] = irrelevant • ISO-on-TCP: local TSAP-ID: – local_tsap_id[1] = B#16#E0; – local_tsap_id[2] = rack and slot of local end points (bits 0 to 4: slot number, bits 5 to 7: rack number); – local_tsap_id[3-16] = TSAP extension, optional • UDP: This parameter is not used. Note: Make sure that every value of local_tsap_id is unique within the CPU.
28 … 33	rem_subnet_id	Array [1..6] of USInt	This parameter is not used.

Byte	Parameter and data type		Description
34 … 39	rem_staddr	Array [1..6] of USInt	TCP and ISO-on-TCP only: IP address of the partner end point. (Not relevant for passive connections.) For example, IP address 192.168.002.003 is stored in the following elements of the array: rem_staddr[1] = 192 rem_staddr[2] = 168 rem_staddr[3] = 002 rem_staddr[4] = 003 rem_staddr[5-6]= irrelevant
40 … 55	rem_tsap_id	Array [1..16] of Byte	Partner address component of connection • TCP: partner port number. Range: 1 to 49151; Recommended values: 2000 to 5000): – rem_tsap_id[1] = high byte of the port number in hexadecimal notation – rem_tsap_id[2] = low byte of the port number in hexadecimal notation; – rem_tsap_id[3-16] = irrelevant • ISO-on-TCP: partner TSAP-ID: – rem_tsap_id[1] = B#16#E0 – rem_tsap_id[2] = rack and slot of partner end point (bits 0 to 4: Slot number, bits 5 to 7: rack number) – rem_tsap_id[3-16] = TSAP extension, optional • UDP: This parameter is not used.
56 … 61	next_staddr	Array [1..6] of Byte	This parameter is not used.
62 … 63	spare	Word	Reserved: W#16#0000

5.6.4 Assigning Internet Protocol (IP) addresses

5.6.4.1 Assigning IP addresses to programming and network devices

If your programming device is using an on-board adapter card connected to your plant LAN (and possibly the world-wide web), the IP Address Network ID and subnet mask of your CPU and the programming device's on-board adapter card must be exactly the same. The Network ID is the first part of the IP address (first three octets) (for example, 211.154.184.16) that determines what IP network you are on. The subnet mask normally has a value of 255.255.255.0; however, since your computer is on a plant LAN, the subnet mask may have various values (for example, 255.255.254.0) in order to set up unique subnets. The subnet mask, when combined with the device IP address in a mathematical AND operation, defines the boundaries of an IP subnet.

Note

In a world-wide web scenario, where your programming devices, network devices, and IP routers will communicate with the world, unique IP addresses must be assigned to avoid conflict with other network users. Contact your company IT department personnel, who are familiar with your plant networks, for assignment of your IP addresses.

If your programming device is using an Ethernet-to-USB adapter card connected to an isolated network, the IP Address Network ID and subnet mask of your CPU and the programming device's Ethernet-to-USB adapter card must be exactly the same. The Network ID is the first part of the IP address (first three octets) (for example, 211.154.184.16) that determines what IP network you are on. The subnet mask normally has a value of 255.255.255.0. The subnet mask, when combined with the device IP address in a mathematical AND operation, defines the boundaries of an IP subnet.

Note

An Ethernet-to-USB adapter card is useful when you do not want your CPU on your company LAN. During initial testing or commissioning tests, this arrangement is particularly useful.

Table 5- 8 Assigning Ethernet addresses

Programming Device Adapter Card	Network Type	Internet Protocol (IP) Address	Subnet Mask
On-board adapter card	Connected to your plant LAN (and possibly the world-wide web)	Network ID of your CPU and the programming device's on-board adapter card must be exactly the same. The Network ID is the first part of the IP address (first three octets) (for example, **211.154.184**.16) that determines what IP network you are on.)	The subnet mask of your CPU and the on-board adapter card must be exactly the same. The subnet mask normally has a value of **255.255.255.0**; however, since your computer is on a plant LAN, the subnet mask may have various values (for example, **255.255.254.0**) in order to set up unique subnets. The subnet mask, when combined with the device IP address in a mathematical AND operation, defines the boundaries of an IP subnet.
Ethernet-to-USB adapter card	Connected to an isolated network	Network ID of your CPU and the programming device's Ethernet-to-USB adapter card must be exactly the same. The Network ID is the first part of the IP address (first three octets) (for example, **211.154.184**.16) that determines what IP network you are on.)	The subnet mask of your CPU and the Ethernet-to-USB adapter card must be exactly the same. The subnet mask normally has a value of **255.255.255.0**. The subnet mask, when combined with the device IP address in a mathematical AND operation, defines the boundaries of an IP subnet.

Assigning or checking the IP address of your programming device using "My Network Places" (on your desktop)

You can assign or check your programming device's IP address with the following menu selections:

- (Right-click) "My Network Places"
- "Properties"
- (Right-click) "Local Area Connection"
- "Properties"

In the "Local Area Connection Properties" dialog, in the "This connection uses the following items:" field, scroll down to "Internet Protocol (TCP/IP)". Click "Internet Protocol (TCP/IP)", and click the "Properties" button. Select "Obtain an IP address automatically (DHCP)" or "Use the following IP address" (to enter a static IP address).

Note

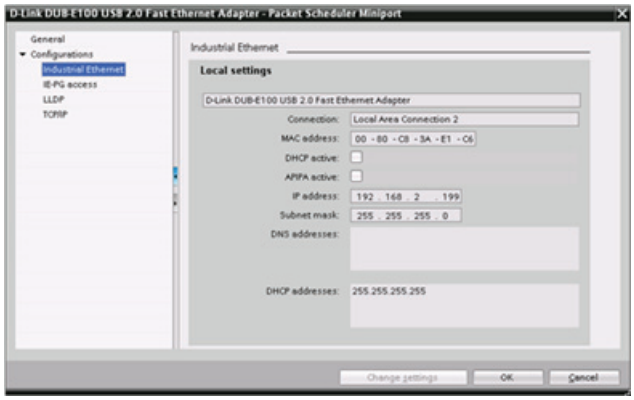
Dynamic Host Configuration Protocol (DHCP) automatically assigns an IP address to your programming device upon power up from the DHCP server.

5.6.4.2 Checking the IP address of your programming device

You can check the MAC and IP addresses of your programming device with the following menu selections:

1. In the "Project tree", expand "Online access".
2. Right-click the required network, and select "Properties".
3. In the network dialog, expand "Configurations", and select "Industrial Ethernet".

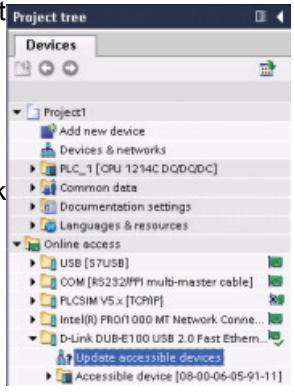
The MAC and IP addresses of the programming device are displayed.



5.6.4.3 Assigning an IP address to a CPU online

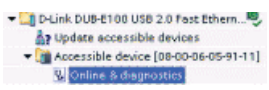
You can assign an IP address to a network device online. This is particularly useful in an initial device configuration.

1. In the "Project tree," verify that no IP address is assigned to the CPU, with the following menu selections:
   - "Online access"
   - <Adapter card for the network in which the device is located>
   - "Update accessible devices"



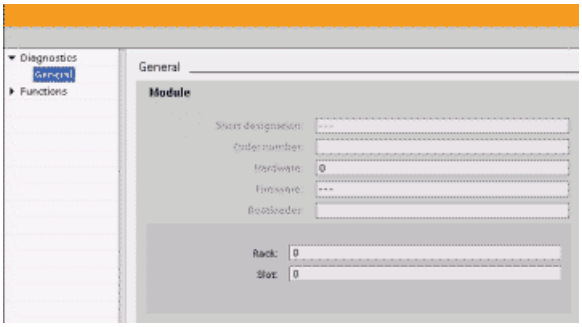
NOTE: If a MAC address is shown instead of an IP address, then no IP address has been assigned.

2. Under the required accessible device, double-click "Online & diagnostics".

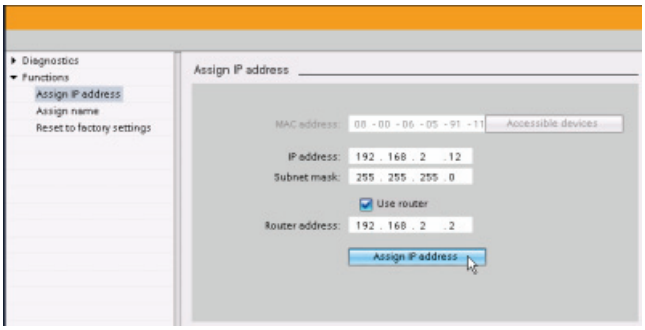


3. In the "Online & diagnostics" dialog, make the following menu selections:

- "Functions"
- "Assign IP address"

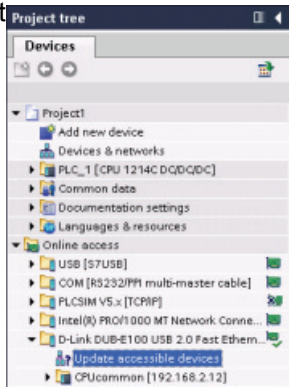


4. In the "IP address" field, enter your new IP address, and click the "Assign IP address" button.



5. In the "Project tree," verify that your new IP address has been assigned to the CPU, with the following menu selections:

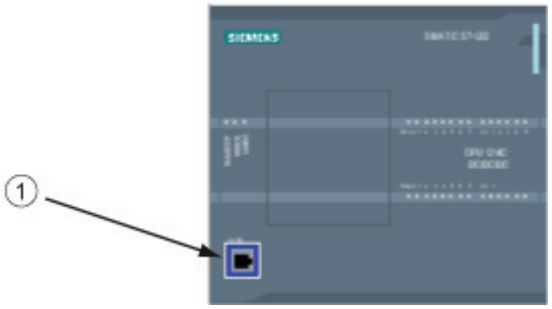
- "Online access"
- <Adapter for the network in which the device is located>
- "Update accessible devices"



5.6.4.4 Configuring an IP address for a CPU in your project

Configuring the PROFINET interface

To configure parameters for the PROFINET interface, select the green PROFINET box on the CPU. The "Properties" tab in the inspector window displays the PROFINET port.



① PROFINET port

Configuring the IP address

Ethernet (MAC) address: In a PROFINET network, each device is assigned a Media Access Control address (MAC address) by the manufacturer for identification. A MAC address consists of six groups of two hexadecimal digits, separated by hyphens (-) or colons (:), in transmission order, (for example, 01-23-45-67-89-AB or 01:23:45:67:89:AB).

IP address: Each device must also have an Internet Protocol (IP) address. This address allows the device to deliver data on a more complex, routed network.

Each IP address is divided into four 8-bit segments and is expressed in a dotted, decimal format (for example, 211.154.184.16). The first part of the IP address is used for the Network ID (What network are you on?), and the second part of the address is for the Host ID (unique for each device on the network). An IP address of 192.168.x.y is a standard designation recognized as part of a private network that is not routed on the Internet.

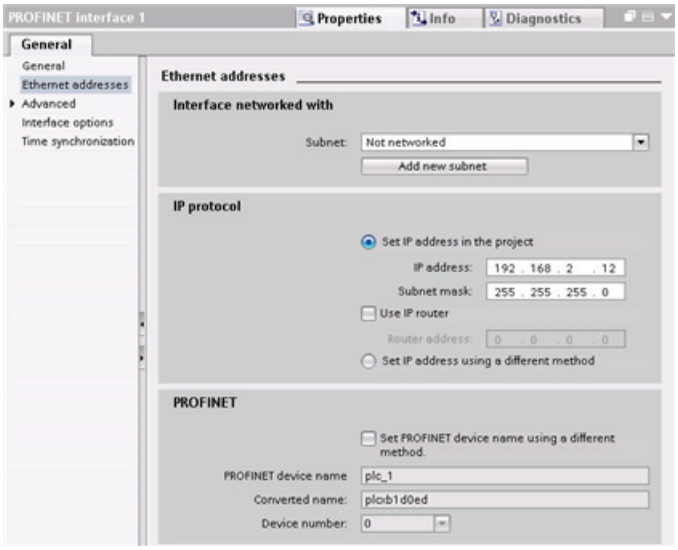
Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet tend to be located in close physical proximity to each other on a Local Area Network (LAN). A mask (known as the subnet mask or network mask) defines the boundaries of an IP subnet.

A subnet mask of 255.255.255.0 is generally suitable for a small local network. This means that all IP addresses on this network should have the same first 3 octets, and the various devices on this network are identified by the last octet (8-bit field). An example of this is to assign a subnet mask of 255.255.255.0 and an IP addresses of 192.168.2.0 through 192.168.2.255 to the devices on a small local network.

The only connection between different subnets is via a router. If subnets are used, an IP router must be employed.

IP router: Routers are the link between LANs. Using a router, a computer in a LAN can send messages to any other networks, which might have other LANs behind them. If the destination of the data is not within the LAN, the router forwards the data to another network or group of networks where it can be delivered to its destination.

Routers rely on IP addresses to deliver and receive data packets.



IP addresses properties: In the Properties window, select the "Ethernet addresses" configuration entry. STEP 7 displays the Ethernet address configuration dialog, which associates the software project with the IP address of the CPU that will receive that project.

Note

All IP addresses are configured when you download the project. If the CPU does not have a pre-configured IP address, you must associate the project with the MAC address of the target device. If your CPU is connected to a router on a network, you must also enter the IP address of the router.

The "Set IP address using a different method" radio button allows you to change the IP address online or by using the "T_CONFIG (Page 421)" instruction after the program is downloaded. This IP address assignment method is for the CPU only.

WARNING

When changing the IP address of a CPU online or from the user program, it is possible to create a condition in which the PROFINET network may stop.

If the IP address of a CPU is changed to an IP address outside the subnet, the PROFINET network will lose communication, and all data exchange will stop. User equipment may be configured to keep running under these conditions. Loss of PROFINET communication may result in unexpected machine or process operations, causing death, severe personal injury, or property damage if proper precautions are not taken.

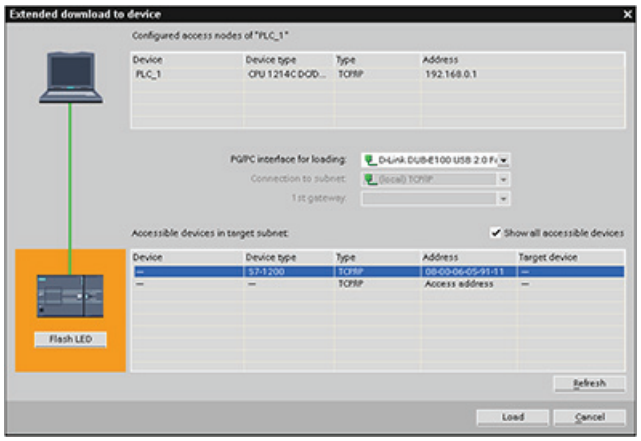
If an IP address must be changed manually, ensure that the new IP address lies within the subnet.

Table 5- 9 Parameters for the IP address

Parameter		Description
Subnet	Name of the Subnet to which the device is connected. Click the "Add new subnet" button to create a new subnet. "Not connected" is the default. Two connection types are possible: • The "Not connected" default provides a local connection. • A subnet is required when your network has two or more devices.	
IP protocol	IP address	Assigned IP address for the CPU
	Subnet mask	Assigned subnet mask
	Use IP router	Click the checkbox to indicate the use of an IP router
	Router address	Assigned IP address for the router, if applicable

5.6.5 Testing the PROFINET network

After completing the configuration, download the project (Page 157) to the CPU. All IP addresses are configured when you download the project.



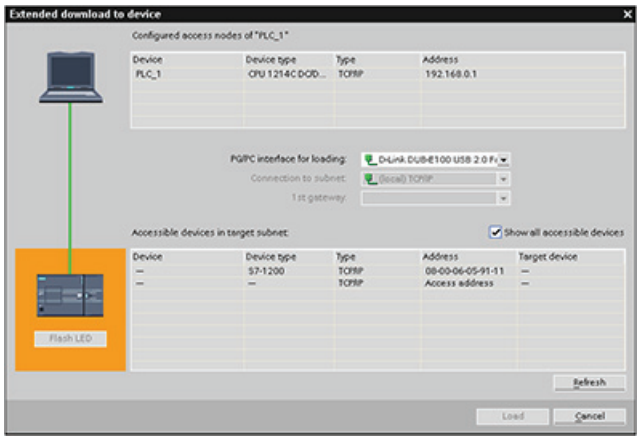
Assigning an IP address to a device online

The S7-1200 CPU does not have a pre-configured IP address. You must manually assign an IP address for the CPU:

- To assign an IP address to a device online, refer to "Device configuration: Assigning an IP address to a CPU online" (Page 125) for this step-by-step procedure.
- To assign an IP address in your project, you must configure the IP address in the Device configuration, save the configuration, and download it to the PLC. Refer to "Device configuration: Configuring an IP address for a CPU in your project" (Page 127) for more information.

Using the "Extended download to device" dialog to test for connected network devices

The S7-1200 CPU "Download to device" function and its "Extended download to device" dialog can show all accessible network devices and whether or not unique IP addresses have been assigned to all devices. To display all accessible and available devices with their assigned MAC or IP addresses, check the "Show all accessible devices" checkbox.



If the required network device is not in this list, communications to that device have been interrupted for some reason. The device and network must be investigated for hardware and/or configuration errors.

5.6.6 Locating the Ethernet (MAC) address on the CPU

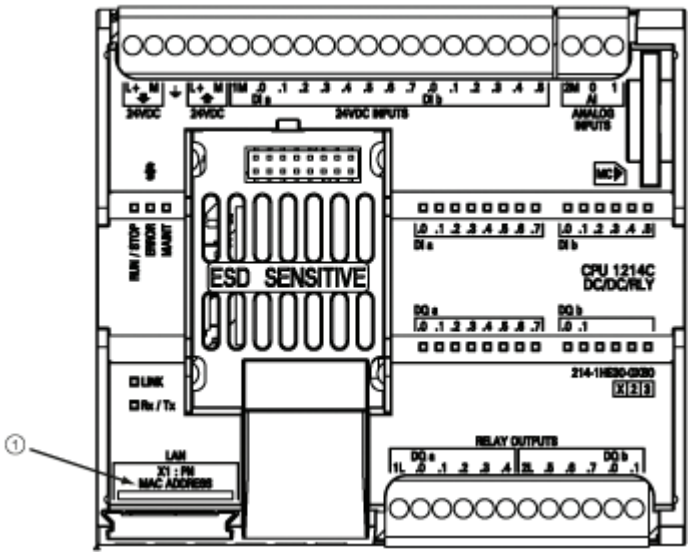
In PROFINET networking, a Media Access Control address (MAC address) is an identifier assigned to the network interface by the manufacturer for identification. A MAC address usually encodes the manufacturer's registered identification number.

The standard (IEEE 802.3) format for printing MAC addresses in human-friendly form is six groups of two hexadecimal digits, separated by hyphens (-) or colons (:), in transmission order, (for example, 01-23-45-67-89-ab or 01:23:45:67:89:ab).

Note

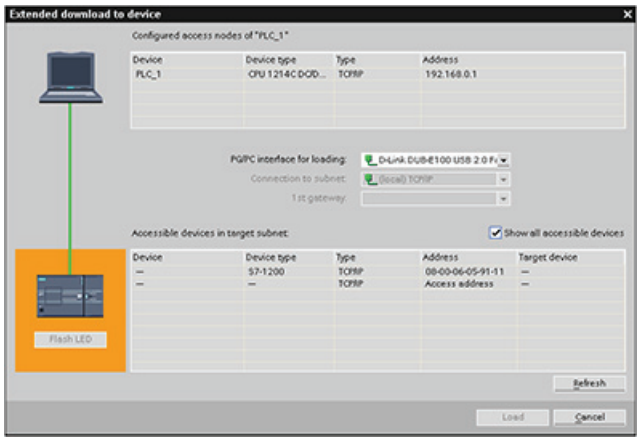
Each CPU is loaded at the factory with a permanent, unique MAC address. You cannot change the MAC address of a CPU.

The MAC address is printed on the front, lower-left corner of the CPU. Note that you have to lift the lower TB doors to see the MAC address information.



① MAC address

Initially, the CPU has no IP address, only a factory-installed MAC address. PROFINET communications requires that all devices be assigned a unique IP address.



Use the CPU "Download to device" function and the "Extended download to device" dialog to show all accessible network devices and ensure that unique IP addresses have been assigned to all devices. This dialog displays all accessible and available devices with their assigned MAC or IP addresses. MAC addresses are all-important in identifying devices that are missing the required unique IP address.

5.6.7 Configuring Network Time Protocol synchronization

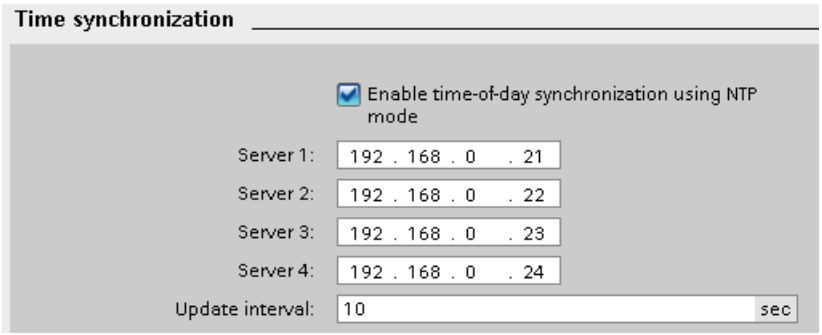
The Network Time Protocol (NTP) is widely used to synchronize the clocks of computer systems to Internet time servers. In NTP mode, the CP sends time-of-day queries at regular intervals (in the client mode) to the NTP server in the subnet (LAN). Based on the replies from the server, the most reliable and most accurate time is calculated and the time of day on the station is synchronized.

The advantage of this mode is that it allows the time to be synchronized across subnets.

The IP addresses of up to four NTP servers need to be configured. The update interval defines the interval between the time queries (in seconds). The value of the interval ranges between 10 seconds and one day.

In NTP mode, it is generally UTC (Universal Time Coordinated) that is transferred; this corresponds to GMT (Greenwich Mean Time).

In the Properties window, select the "Time synchronization" configuration entry. STEP 7 displays the Time synchronization configuration dialog:



Note

All IP addresses are configured when you download the project.

Table 5- 10 Parameters for time synchronization

Parameter	Definition
Enable time-of-day synchronization using Network Time Protocol (NTP) servers	Click the checkbox to enable time-of-day synchronization using NTP servers.
Server 1	Assigned IP Address for network time server 1
Server 2	Assigned IP Address for network time server 2
Server 3	Assigned IP Address for network time server 3
Server 4	Assigned IP Address for network time server 4
Time synchronization interval	Interval value (sec)

5.6.8 PROFINET device start-up time, naming, and address assignment

PROFINET IO can extend the start-up time for your system (configurable time-out figure). More devices and slow devices impact the amount of time it takes to switch to RUN. There is a maximum of 8 IO devices on the S7-1200 PROFINET network.

Each station (or IO device) starts up independently on start-up, and this affects the overall CPU start-up time. If you set the configurable time-out too low, there may not be a sufficient overall CPU start-up time for all stations to complete start-up. If this situation occurs, false station errors will result.

The default configurable time-out is 1 minute; the user can configure this time.

PROFINET device naming and addressing in STEP 7

All PROFINET devices **must** have a Device Name and an IP Address. Use STEP 7 to define the Device Names and to configure the IP addresses. Device names are downloaded to the IO devices using PROFINET DCP (Discovery and Configuration Protocol).

PROFINET address assignment at system start-up

The controller broadcasts the names of the devices to the network, and the devices respond with their MAC addresses. The controller then assigns an IP address to the device using PROFINET DCP protocol:

- If the MAC address has a configured IP address, then the station performs start-up.
- If the MAC address does not have a configured IP address, STEP 7 assigns the address that is configured in the project, and the station then performs start-up.
- If there is a problem with this process, a station error occurs and no start-up takes place. This situation causes the configurable time-out value to be exceeded.

Programming concepts

6

6.1 Guidelines for designing a PLC system

When designing a PLC system, you can choose from a variety of methods and criteria. The following general guidelines can apply to many design projects. Of course, you must follow the directives of your own company's procedures and the accepted practices of your own training and location.

Table 6- 1 Guidelines for designing a PLC system

Recommended steps	Tasks
Partition your process or machine	Divide your process or machine into sections that have a level of independence from each other. These partitions determine the boundaries between controllers and influence the functional description specifications and the assignment of resources.
Create the functional specifications	Write the descriptions of operation for each section of the process or machine, such as the I/O points, the functional description of the operation, the states that must be achieved before allowing action for each actuator (such as a solenoid, a motor, or a drive), a description of the operator interface, and any interfaces with other sections of the process or machine.
Design the safety circuits	Identify any equipment that might require hard-wired logic for safety. Remember that control devices can fail in an unsafe manner, which can produce unexpected startup or change in the operation of machinery. Where unexpected or incorrect operation of the machinery could result in physical injury to people or significant property damage, consider the implementation of electromechanical overrides (which operate independently of the PLC) to prevent unsafe operations. The following tasks should be included in the design of safety circuits: • Identify any improper or unexpected operation of actuators that could be hazardous. • Identify the conditions that would assure the operation is not hazardous, and determine how to detect these conditions independently of the PLC. • Identify how the PLC affects the process when power is applied and removed, and also identify how and when errors are detected. Use this information only for designing the normal and expected abnormal operation. You should not rely on this "best case" scenario for safety purposes. • Design the manual or electromechanical safety overrides that block the hazardous operation independent of the PLC. • Provide the appropriate status information from the independent circuits to the PLC so that the program and any operator interfaces have necessary information. • Identify any other safety-related requirements for safe operation of the process.
Specify the operator stations	Based on the requirements of the functional specifications, create the following drawings of the operator stations: • Overview drawing that shows the location of each operator station in relation to the process or machine. • Mechanical layout drawing of the devices for the operator station, such as display, switches, and lights. • Electrical drawings with the associated I/O of the PLC and signal modules.

Recommended steps	Tasks
Create the configuration drawings	Based on the requirements of the functional specification, create configuration drawings of the control equipment: • Overview drawing that shows the location of each PLC in relation to the process or machine. • Mechanical layout drawing of each PLC and any I/O modules, including any cabinets and other equipment. • Electrical drawings for each PLC and any I/O modules, including the device model numbers, communications addresses, and I/O addresses.
Create a list of symbolic names	Create a list of symbolic names for the absolute addresses. Include not only the physical I/O signals, but also the other elements (such as tag names) to be used in your program.

6.2 Structuring your user program

When you create a user program for the automation tasks, you insert the instructions for the program into code blocks:

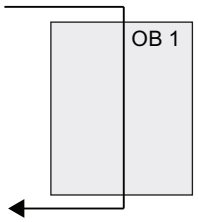
- An organization block (OB) responds to a specific event in the CPU and can interrupt the execution of the user program. The default for the cyclic execution of the user program (OB 1) provides the base structure for your user program and is the only code block required for a user program. If you include other OBs in your program, these OBs interrupt the execution of OB 1. The other OBs perform specific functions, such as for startup tasks, for handling interrupts and errors, or for executing specific program code at specific time intervals.
- A function block (FB) is a subroutine that is executed when called from another code block (OB, FB, or FC). The calling block passes parameters to the FB and also identifies a specific data block (DB) that stores the data for the specific call or instance of that FB. Changing the instance DB allows a generic FB to control the operation of a set of devices. For example, one FB can control several pumps or valves, with different instance DBs containing the specific operational parameters for each pump or valve.
- A function (FC) is a subroutine that is executed when called from another code block (OB, FB, or FC). The FC does not have an associated instance DB. The calling block passes parameters to the FC. The output values from the FC must be written to a memory address or to a global DB.

Choosing the type of structure for your user program

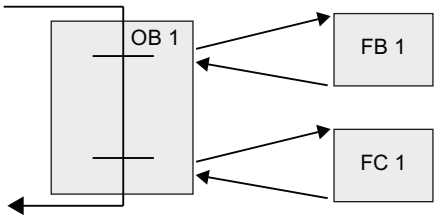
Based on the requirements of your application, you can choose either a linear structure or a modular structure for creating your user program:

- A linear program executes all of the instructions for your automation tasks in sequence, one after the other. Typically, the linear program puts all of the program instructions into the OB for the cyclic execution of the program (OB 1).
- A modular program calls specific code blocks that perform specific tasks. To create a modular structure, you divide the complex automation task into smaller subordinate tasks that correspond to the technological functions of the process. Each code block provides the program segment for each subordinate task. You structure your program by calling one of the code blocks from another block.

Linear structure:



Modular structure:



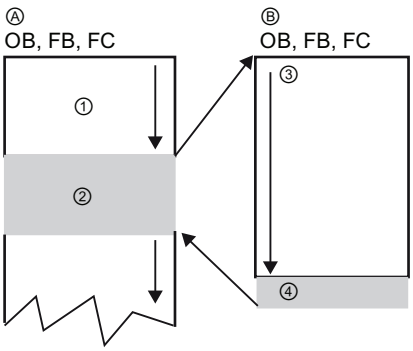
By creating generic code blocks that can be reused within the user program, you can simplify the design and implementation of the user program. Using generic code blocks has a number of benefits:

- You can create reusable blocks of code for standard tasks, such as for controlling a pump or a motor. You can also store these generic code blocks in a library that can be used by different applications or solutions.
- When you structure the user program into modular components that relate to functional tasks, the design of your program can be easier to understand and to manage. The modular components not only help to standardize the program design, but can also help to make updating or modifying the program code quicker and easier.
- Creating modular components simplifies the debugging of your program. By structuring the complete program as a set of modular program segments, you can test the functionality of each code block as it is developed.
- Creating modular components that relate to specific technological functions can help to simplify and reduce the time involved with commissioning the completed application.

6.3 Using blocks to structure your program

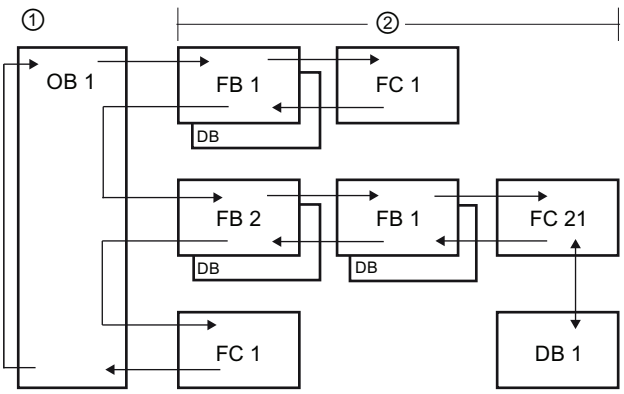
By designing FBs and FCs to perform generic tasks, you create modular code blocks. You then structure your program by having other code blocks call these reusable modules. The calling block passes device-specific parameters to the called block.

When a code block calls another code block, the CPU executes the program code in the called block. After execution of the called block is complete, the CPU resumes the execution of the calling block. Processing continues with execution of the instruction that follows after the block call.



A Calling block
B Called (or interrupting) block
① Program execution
② Instruction or event that initiates the execution of another block
③ Program execution
④ Block end (returns to calling block)

You can nest the block calls for a more modular structure. In the following example, the nesting depth is 4: the program cycle OB plus 3 layers of calls to code blocks.



① Start of cycle
② Nesting depth

6.3.1 Organization block (OB)

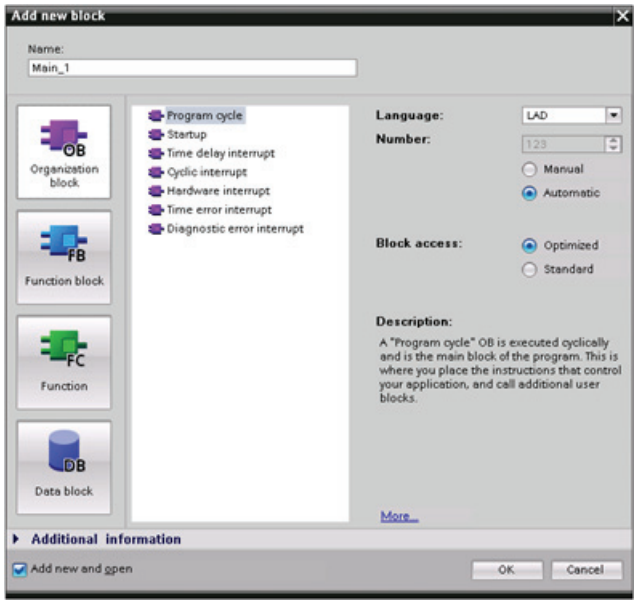
Organization blocks provide structure for your program. They serve as the interface between the operating system and the user program. OBs are event driven. An event, such as a diagnostic interrupt or a time interval, will cause the CPU to execute an OB. Some OBs have predefined start events and behavior.

The program cycle OB contains your main program. You can include more than one program cycle OB in your user program. During RUN mode, the program cycle OBs execute at the lowest priority level and can be interrupted by all other types of program processing. The startup OB does not interrupt the program cycle OB because the CPU executes the startup OB before going to RUN mode.

After finishing the processing of the program cycle OBs, the CPU immediately executes the program cycle OBs again. This cyclic processing is the "normal" type of processing used for programmable logic controllers. For many applications, the entire user program is located in a single program cycle OB.

You can create other OBs to perform specific functions, such as for handling interrupts and errors, or for executing specific program code at specific time intervals. These OBs interrupt the execution of the program cycle OBs.

Use the "Add new block" dialog to create new OBs in your user program.



Interrupt handling is always event-driven. When such an event occurs, the CPU interrupts the execution of the user program and calls the OB that was configured to handle that event. After finishing the execution of the interrupting OB, the CPU resumes the execution of the user program at the point of interruption.

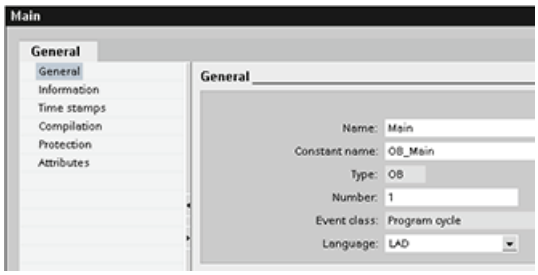
The CPU determines the order for handling interrupt events by a priority assigned to each OB. Each event has a particular servicing priority. The respective priority level within a priority class determines the order in which the OBs are executed. Several interrupt events can be combined into priority classes. For more information, refer to the PLC concepts chapter section on execution of the user program (Page 61).

Creating an additional OB within a class of OB

You can create multiple OBs for your user program, even for the program cycle and startup OB classes. Use the "Add new block" dialog to create an OB. Enter the name for your OB and enter an OB number 200 or greater.

If you create multiple program cycle OBs for your user program, the CPU executes each program cycle OB in numerical sequence, starting with the program cycle OB with the lowest number (such as OB 1). For example: after first program cycle OB (such as OB1) finishes, the CPU executes the next higher program cycle OB (such as OB 200).

Configuring the operation of an OB



You can modify the operational parameters for an OB. For example, you can configure the time parameter for a time-delay OB or for a cyclic OB.

6.3.2 Function (FC)

A function (FC) is a code block that typically performs a specific operation on a set of input values. The FC stores the results of this operation in memory locations. For example, use FCs to perform standard and reusable operations (such as for mathematical calculations) or technological functions (such as for individual controls using bit logic operations). An FC can also be called several times at different points in a program. This reuse simplifies the programming of frequently recurring tasks.

An FC does not have an associated instance data block (DB). The FC uses the local data stack for the temporary data used to calculate the operation. The temporary data is not saved. To store data permanently, assign the output value to a global memory location, such as M memory or to a global DB.

6.3.3 Function block (FB)

A function block (FB) is a code block that uses an instance data block for its parameters and static data. FBs have variable memory that is located in a data block (DB), or "instance" DB. The instance DB provides a block of memory that is associated with that instance (or call) of the FB and stores data after the FB finishes. You can associate different instance DBs with different calls of the FB. The instance DBs allow you to use one generic FB to control multiple devices. You structure your program by having one code block make a call to an FB and an instance DB. The CPU then executes the program code in that FB, and stores the block parameters and the static local data in the instance DB. When the execution of the FB finishes, the CPU returns to the code block that called the FB. The instance DB retains the values for that instance of the FB. These values are available to subsequent calls to the function block either in the same scan cycle or other scan cycles.

Reusable code blocks with associated memory

You typically use an FB to control the operation for tasks or devices that do not finish their operation within one scan cycle. To store the operating parameters so that they can be quickly accessed from one scan to the next, each FB in your user program has one or more instance DBs. When you call an FB, you also specify an instance DB that contains the block parameters and the static local data for that call or "instance" of the FB. The instance DB maintains these values after the FB finishes execution.

By designing the FB for generic control tasks, you can reuse the FB for multiple devices by selecting different instance DBs for different calls of the FB.

An FB stores the Input, Output, and InOut, and Static parameters in an instance DB.

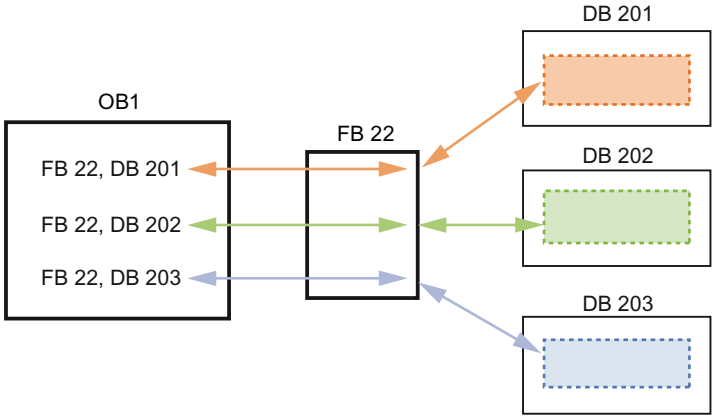
Assigning the start value in the instance DB

The instance DB stores both a default value and a start value for each parameter. The start value provides the value to be used when the FB is executed. The start value can then be modified during the execution of your user program.

The FB interface also provides a "Default value" column that allows you to assign a new start value for the parameter as you are writing the program code. This default value in the FB is then transferred to the start value in the associated instance DB. If you do not assign a new start value for a parameter in the FB interface, the default value from instance DB is copied to start value.

Using a single FB with DBs

The following figure shows an OB that calls one FB three times, using a different data block for each call. This structure allows one generic FB to control several similar devices, such as motors, by assigning a different instance data block for each call for the different devices. Each instance DB stores the data (such as speed, ramp-up time, and total operating time) for an individual device.



In this example, FB 22 controls three separate devices, with DB 201 storing the operational data for the first device, DB 202 storing the operational data for the second device, and DB 203 storing the operational data for the third device.

6.3.4 Data block (DB)

You create data blocks (DB) in your user program to store data for the code blocks. All of the program blocks in the user program can access the data in a global DB, but an instance DB stores data for a specific function block (FB).

The data stored in a DB is not deleted when the execution of the associated code block comes to an end. There are two types of DBs:

- A global DB stores data for the code blocks in your program. Any OB, FB, or FC can access the data in a global DB.
- An instance DB stores the data for a specific FB. The structure of the data in an instance DB reflects the parameters (Input, Output, and InOut) and the static data for the FB. (The Temp memory for the FB is not stored in the instance DB.)

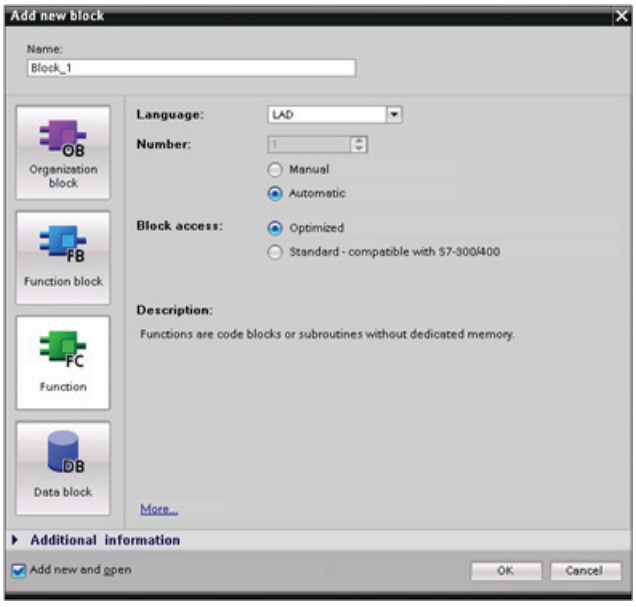
Note

Although the instance DB reflects the data for a specific FB, any code block can access the data in an instance DB.

You can configure a DB as being read-only:

1. Right-click the DB in the project navigator and select "Properties" from the context menu.
2. In the "Properties" dialog, select "Attributes".
3. Select the "Data block write-protected in the device" option and click "OK".

Creating reusable code blocks



Use the "Add new block" dialog under "Program blocks" in the Project navigator to create OBs, FBs, FCs, and global DBs.

When you create a code block, you select the programming language for the block. You do not select a language for a DB because it only stores data.

6.4 Understanding data consistency

The CPU maintains the data consistency for all of the elementary data types (such as Words or DWords) and all of the system-defined structures (for example, IEC_TIMERS or DTL). The reading or writing of the value cannot be interrupted. (For example, the CPU protects the access to a DWord value until the four bytes of the DWord have been read or written.) To ensure that the program cycle OBs and the interrupt OBs cannot write to the same memory location at the same time, the CPU does not execute an interrupt OB until the read or write operation in the program cycle OB has been completed.

If your user program shares multiple values in memory between a program cycle OB and an interrupt OB, your user program must also ensure that these values are modified or read consistently. You can use the DIS_AIRT (disable alarm interrupt) and EN_AIRT (enable alarm interrupt) instructions in your program cycle OB to protect any access to the shared values.

- Insert a DIS_AIRT instruction in the code block to ensure that an interrupt OB cannot be executed during the read or write operation.
- Insert the instructions that read or write the values that could be altered by an interrupt OB.
- Insert an EN_AIRT instruction at the end of the sequence to cancel the DIS_AIRT and allow the execution of the interrupt OB.

A communication request from an HMI device or another CPU can also interrupt execution of the program cycle OB. The communication requests can also cause issues with data consistency. The CPU ensures that the elementary data types are always read and written consistently by the user program instructions. Because the user program is interrupted periodically by communications, it is not possible to guarantee that multiple values in the CPU will all be updated at the same time by the HMI. For example, the values displayed on a given HMI screen could be from different scan cycles of the CPU.

The PtP (Point-to-Point) instructions, PROFINET instructions (such as TSEND_C and TRCV_C), PROFINET Distributed I/O instructions (Page 441) , and PROFIBUS Distributed I/O Instructions (Page 451) transfer buffers of data that could be interrupted. Ensure the data consistency for the buffers of data by avoiding any read or write operation to the buffers in both the program cycle OB and an interrupt OB. If it is necessary to modify the buffer values for these instructions in an interrupt OB, use a DIS_AIRT instruction to delay any interruption (an interrupt OB or a communication interrupt from an HMI or another CPU) until an EN_AIRT instruction is executed.

Note

The use of the DIS_AIRT instruction delays the processing of interrupt OBs until the EN_AIRT instruction is executed, affecting the interrupt latency (time from an event to the time when the interrupt OB is executed) of your user program.

6.5 Programming language

STEP 7 provides the following standard programming languages for S7-1200:

- LAD (ladder logic) is a graphical programming language. The representation is based on circuit diagrams (Page 144).
- FBD (Function Block Diagram) is a programming language that is based on the graphical logic symbols used in Boolean algebra (Page 145).
- SCL (structured control language) is a text-based, high-level programming language (Page 145).

When you create a code block, you select the programming language to be used by that block.

Your user program can utilize code blocks created in any or all of the programming languages.

6.5.1 Ladder logic (LAD)

The elements of a circuit diagram, such as normally closed and normally open contacts, and coils are linked to form networks.

To create the logic for complex operations, you can insert branches to create the logic for parallel circuits. Parallel branches are opened downwards or are connected directly to the power rail. You terminate the branches upwards.

LAD provides "box" instructions for a variety of functions, such as math, timer, counter, and move.

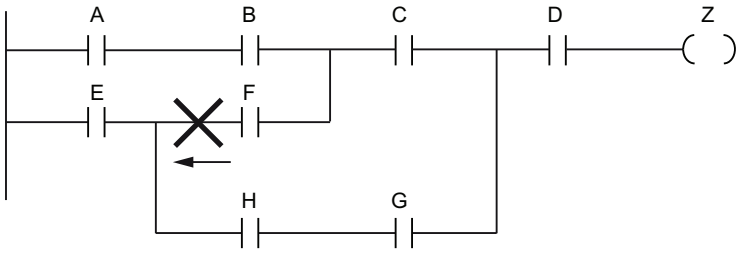
STEP 7 does not limit the number of instructions (rows and columns) in a LAD network.

Note

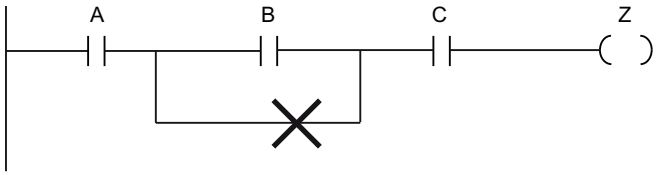
Every LAD network must terminate with a coil or a box instruction.

Consider the following rules when creating a LAD network:

- You cannot create a branch that could result in a power flow in the reverse direction.

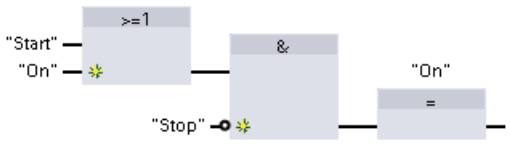


- You cannot create a branch that would cause a short circuit.



6.5.2 Function Block Diagram (FBD)

Like LAD, FBD is also a graphical programming language. The representation of the logic is based on the graphical logic symbols used in Boolean algebra.



To create the logic for complex operations, insert parallel branches between the boxes.

Mathematical functions and other complex functions can be represented directly in conjunction with the logic boxes.

STEP 7 does not limit the number of instructions (rows and columns) in an FBD network.

6.5.3 SCL

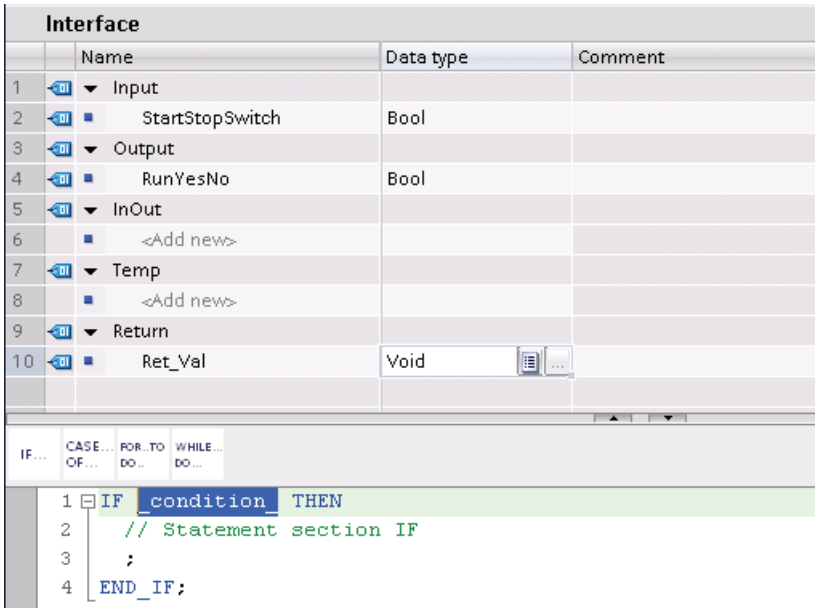
Structured Control Language (SCL) is a high-level, PASCAL-based programming language for the SIMATIC S7 CPUs. SCL supports the block structure of STEP 7 (Page 138). You can also include program blocks written in SCL with program blocks written in LAD and FBD.

SCL instructions use standard programming operators, such as for assignment (:=), mathematical functions (+ for addition, - for subtraction, * for multiplication, and / for division). SCL also uses standard PASCAL program control operations, such as IF-THEN-ELSE, CASE, REPEAT-UNTIL, GOTO and RETURN. You can use any PASCAL reference for syntactical elements of the SCL programming language. Many of the other instructions for SCL, such as timers and counters, match the LAD and FBD instructions. For more information about specific instructions, refer to the specific instructions in the chapters for Basic instructions (Page 163) and Extended instructions (Page 233).

You can designate any type of block (OB, FB, or FC) to use the SCL programming language at the time you create the block. STEP 7 provides an SCL program editor that includes the following elements:

- Interface section for defining the parameters of the code block
- Code section for the program code
- Instruction tree that contains the SCL instructions supported by the CPU

You enter the SCL code for your instruction directly in the code section. For more complex instructions, simply drag the SCL instructions from the instruction tree and drop them into your program. You can also use any text editor to create an SCL program and then import that file into STEP 7.



In the section of the SCL code block you can declare the following types of parameters:

- Input, Output, InOut, and Ret_Val: These parameters define the input tags, output tags, and return value for the code block. The tag name that you enter here is used locally during the execution of the code block. You typically would not use the global tag name in the tag table.
- Static (FBs only; the illustration above is for an FC): Static tags are used for storage of static intermediate results in the instance data block. Static data is retained until overwritten, which may be after several cycles. The names of the blocks, which are called in this code block as multi-instance, are also stored in the static local data.
- Temp: These parameters are the temporary tags that are used during the execution of the code block.

If you call the SCL code block from another code block, the parameters of the SCL code block appear as inputs or outputs.

In this example, the tags for "Start" and "On" (from the project tag table) correspond to "StartStopSwitch" and "RunYesNo" in the declaration table of the SCL program.

Constructing an SCL expression

An SCL expression is a formula for calculating a value. The expression consists of operands and operators (such as *, /, + or -). The operands can be tags, constants, or expressions.

The evaluation of the expression occurs in a certain order, which is defined by the following factors:

- Every operator has a pre-defined priority, with the highest-priority operation performed first.
- For operators with equal priority, the operators are processed in a left-to-right sequence.
- You use parentheses to designate a series of operators to be evaluated together.

The result of an expression can be used either for assigning a value to a tag used by your program, as a condition to be used by a control statement, or as parameters for another SCL instruction or for calling a code block.

Table 6- 2 Operators in SCL

Type	Operation	Operator	Priority
Parentheses	(*Expression*)	(,)	1
Math	Power	**	2
	Sign (unary plus)	+	3
	Sign (unary minus)	-	3
	Multiplication	*	4
	Division	/ or DIV	4

Type	Operation	Operator	Priority
	Modulo	MOD	4
	Addition	+	5
	Subtraction	-	5
Comparison	Less than	<	6
	Less than or equal to	<=	6
	Greater than	>	6
	Greater than or equal to	>=	6
	Equal to	=	7
	Not equal to	<>	7
Bit logic	Negation (unary)	NOT	3
	AND logic operation	AND or &	8
	Exclusive OR logic operation	XOR	9
	OR logic operation	OR	10
Assignment	Assignment	:=	11

As a high-level programming language, SCL uses standard statements for basic tasks:

- Assignment statement: :=
- Mathematical functions: +, -, *, and /
- Addressing of global variables (tags): "<tag name>" (Tag name or data block name enclosed in double quotes)
- Addressing of local variables: #<variable name> (Variable name preceded by "#" symbol)

The following examples show different expressions for different uses.

`"C" := #A+#B;`	Assigns the sum of two local variables to a tag
`"Data_block_1".Tag := #A;`	Assignment to a data block tag
`IF #A > #B THEN "C" := #A;`	Condition for the IF-THEN statement
`"C" := SQRT (SQR (#A) + SQR (#B));`	Parameters for the SQRT instruction

Arithmetic operators can process various numeric data types. The data type of the result is determined by the data type of the most-significant operands. For example, a multiplication operation that uses an INT operand and a REAL operand yields a REAL value for the result.

Control statements

A control statement is a specialized type of SCL expression that performs the following tasks:

- Program branching
- Repeating sections of the SCL program code
- Jumping to other parts of the SCL program
- Conditional execution

The SCL control statements include IF-THEN, CASE-OF, FOR-TO-DO, WHILE-DO, REPEAT-UNTIL, CONTINUE, GOTO, and RETURN.

A single statement typically occupies one line of code. You can enter multiple statements on one line, or you can break a statement into several lines of code to make the code easier to read. Separators (such as tabs, line breaks and extra spaces) are ignored during the syntax check. An END statement terminates the control statement.

The following examples show a FOR-TO-DO control statement. (Both forms of coding are syntactically valid.)

```
FOR x := 0 TO max DO sum := sum + value(x); END_FOR;
FOR x := 0 TO max DO
     sum := sum + value(x);
END_FOR;
```

A control statement can also be provided with a label. A label is set off by a colon at the beginning of the statement:

```
Label: <Statement>;
```

The STEP 7 online help provides a complete SCL programming language reference.

Conditions

A condition is a comparison expression or a logical expression whose result is of type BOOL (with the value of either TRUE or FALSE). The following example shows conditions of various types.

`#Temperature > 50`	Relational expression
`#Counter <= 100`	
`#CHAR1 < 'S'`	
`(#Alpha <> 12) AND NOT #Beta`	Comparison and logical expression
`5 + #Alpha`	Arithmetic expression

A condition can use arithmetic expressions:

- The condition of the expression is TRUE if the result is any value other than zero.
- The condition of the expression is FALSE if the result equals zero.

Addressing

As with LAD and FBD, SCL allows you to use either tags (symbolic addressing) or absolute addresses in your user program. SCL also allows you to use a variable as an array index.

Absolute addressing

```
I[byteindex.bitindex]
MB[byteindex]
```

In these examples, the tags "byteindex" and "bitindex" are tags that store the value (Int) used to determine the address to be accessed. Your user program can change the value of these tags, which then changes the address in the statement or expression.

Symbolic addressing	
`"PLC_Tag_1"`	Tag in PLC tag table
`"Data_block_1".Tag_1`	Tag in a data block
`"Data_block_1".MyArray[#i]`	Array element in a data block array

Indexed addressing with PEEK and POKE instructions

SCL also provides PEEK and POKE instructions to allow you to read from or write to data areas with variables that provide specific byte offsets or bit offsets for the operation.

```
PEEK(area:=_in_,
          dbNumber:=_in_,
          byteOffset:=_in_);
```

Reads the byte referenced by byteOffset of the referenced data block, I/O or memory area.

```
Example: MB100 := PEEK(area:=16#84,
dbNumber:=1, byteOffset:=#i);
```

```
PEEK_WORD(area:=_in_,
          dbNumber:=_in_,
          byteOffset:=_in_);
```

Reads the word referenced by byteOffset of the referenced data block, I/O or memory area.

```
Example: MW200 := PEEK(area:=16#84,
dbNumber:=1, byteOffset:=#i);
```

```
PEEK_DWORD(area:=_in_,
          dbNumber:=_in_,
          byteOffset:=_in_);
```

Reads the double word referenced by byteOffset of the referenced data block, I/O or memory area.

```
Example: MW200 := PEEK(area:=16#84,
dbNumber:=1, byteOffset:=#i);
```

```
PEEK_BOOL(area:=_in_,
          dbNumber:=_in_,
          byteOffset:=_in_,
          bitOffset:=_in_);
```

Reads a Boolean referenced by the bitOffset and byteOffset of the referenced data block, I/O or memory area

Example: MB100.0 := PEEK(area:=16#84, dbNumber:=1, byteOffset:=#ii, bitOffset:=#j);

```
POKE(area:=_in_,
          dbNumber:=_in_,
          byteOffset:=_in_,
          value:=_in_);
```

Writes the value to the referenced byteOffset of the referenced data block, I/O or memory area

Example: POKE(area:=16#84, dbNumber:=2, byteOffset:=3, value:="Tag_1");

```
POKE_BOOL(area:=_in_,
          dbNumber:=_in_,
          byteOffset:=_in_,
          bitOffset:=_in_,
          value:=_in_);
```

Writes the Boolean value to the referenced bitOffset and byteOffset of the referenced data block, I/O or memory area

Example: POKE_BOOL(area:=16#84, dbNumber:=2, byteOffset:=3, bitOffset:=5, value:=0);

```
POKE_BLK(area_src:=_in_,
          dbNumber_src:=_in_,
          byteOffset_src:=_in_,
          area_dest:=_in_,
          dbNumber_dest:=_in_,
          byteOffset_dest:=_in_,
          count:=_in_);
```

Writes "count" number of bytes starting at the referenced byte Offset of the referenced source data block, I/O or memory area to the referenced byteOffset of the referenced destination data block, I/O or memory area

Example: POKE_BLK(area_src:=16#84, dbNumber_src:=#src_db, byteOffset_src:=#src_byte, area_dest:=16#84, dbNumber_dest:=#src_db, byteOffset_dest:=#src_byte, count:=10);

For PEEK and POKE instructions, the following values for the "area", "area_src" and "area_dest" parameters are applicable. For areas other than data blocks, the dbNumber parameter must be 0.

16#1	PI
16#2	PQ
16#81	I
16#82	Q
16#83	M
16#84	DB

Calling other code blocks from your SCL program

To call another code block in your user program, simply enter the name (or absolute address) of the FB or FC with the parameters. For an FB, you must provide the instance DB to be called with the FB.

```
<DB name> (Parameter list)
```

Call as a single instance

```
<#Instance name> (Parameter list)
```

Call as multi-instance

```
"MyDB"(MyInput:=10, MyInOut:="Tag1");
```

```
<FC name> (Parameter list)
```

Standard call

```
<Operand>:=<FC name> (Parameter list)
```

Call in an expression

```
"MyFC"(MyInput:=10, MyInOut:="Tag1");
```

You can also drag blocks from the navigation tree to the SCL program editor, and complete the parameter assignment.

6.5.4 EN and ENO for LAD, FBD and SCL

Determining "power flow" (EN and ENO) for an instruction

Certain instructions (such as the Math and the Move instructions) provide parameters for EN and ENO. These parameters relate to power flow in LAD or FBD and determine whether the instruction is executed during that scan. SCL also allows you to set the ENO parameter for a code block.

- EN (Enable In) is a Boolean input. Power flow (EN = 1) must be present at this input for the box instruction to be executed. If the EN input of a LAD box is connected directly to the left power rail, the instruction will always be executed.
- ENO (Enable Out) is a Boolean output. If the box has power flow at the EN input and the box executes its function without error, then the ENO output passes power flow (ENO = 1) to the next element. If an error is detected in the execution of the box instruction, then power flow is terminated (ENO = 0) at the box instruction that generated the error.

Table 6- 3 Operands for EN and ENO

Program editor	Inputs/outputs	Operands	Data type
LAD	EN, ENO	Power flow	Bool
FBD	EN	I, I:P, Q, M, DB, Temp, Power Flow	Bool
	ENO	Power Flow	Bool
SCL	EN[1]	TRUE, FALSE	Bool
	ENO[2]	TRUE, FALSE	Bool

1 The use of EN is only available for FBs.

2. The use of ENO with the SCL code block is optional. You must configure the SCL compiler to set ENO when the code block finishes.

Configuring SCL to set ENO

For checking the operation of individual statements within the SCL code, use OK or NOT OK (Page 185) with an IF-THEN or other conditional construction.

To configure the SCL compiler for setting ENO, follow these steps:

1. Select the "Settings" command from the "Options" menu.

2. Expand the "PLC programming" properties and select "SCL (Structured Control Language)".

3. Select the "Set ENO automatically" option.

Effect of Ret_Val or Status parameters on ENO

Some instructions, such as the communication instructions or the string conversion instructions, provide an output parameter that contains information about the processing of the instruction. For example, some instructions provide a Ret_Val (return value) parameter, which is typically an Int data type that contains status information in a range from -32768 to +32767. Other instructions provide a Status parameter, which is typically a Word data type that stores status information in a range of hexadecimal values from 16#0000 to 16#FFFF. The numerical value stored in a Ret_Val or a Status parameter determines the state of ENO for that instruction.

- Ret_Val: A value from 0 to 32767 typically sets ENO = 1 (or TRUE). A value from -32768 to -1 typically sets ENO = 0 (or FALSE). To evaluate Ret_Val, change the representation to hexadecimal.
- Status: A value from 16#0000 16#7FFF typically sets ENO = 1 (or TRUE). A value from 16#8000 to 16#FFFF typically sets ENO = 0 (or FALSE).

Instructions that take more than one scan to execute often provide a Busy parameter (Bool) to signal that the instruction is active but has not completed execution. These instructions often also provide a Done parameter (Bool) and an Error parameter (Bool). Done signals that the instruction was completed without error, and Error signals that the instruction was completed with an error condition.

- When Busy = 1 (or TRUE), ENO = 1 (or TRUE).
- When Done = 1 (or TRUE), ENO = 1 (or TRUE).
- When Error = 1 (or TRUE), ENO = 0 (or FALSE).

6.6 Protection

6.6.1 Access protection for the CPU

The CPU provides 3 levels of security for restricting access to specific functions. When you configure the security level and password for a CPU, you limit the functions and memory areas that can be accessed without entering a password.

The password is case-sensitive.

To configure the password, follow these steps:

1. In the "Device configuration", select the CPU.
2. In the inspector window, select the "Properties" tab.
3. Select the "Protection" property to select the protection level and to enter a password.

Each level allows certain functions to be accessible without a password. The default condition for the CPU is to have no restriction and no password-protection. To restrict access to a CPU, you configure the properties of the CPU and enter the password.

Entering the password over a network does not compromise the password protection for the CPU. A password-protected CPU allows only one user unrestricted access at a time. Password protection does not apply to the execution of user program instructions including communication functions. Entering the correct password provides access to all of the functions.

PLC-to-PLC communications (using communication instructions in the code blocks) are not restricted by the security level in the CPU. HMI functionality is also not restricted.

Table 6- 4 Security levels for the CPU

Security level	Access restrictions
No protection	Allows full access without password-protection.
Write protection	Allows HMI access and all forms of PLC-to-PLC communications without password-protection. Password is required for modifying (writing to) the CPU and for changing the CPU mode (RUN/STOP).
Read/write protection	Allows HMI access and all forms of PLC-to-PLC communications without password-protection. Password is required for reading the data in the CPU, for modifying (writing to) the CPU, and for changing the CPU mode (RUN/STOP).

6.6.2 Know-how protection

Know-how protection allows you to prevent one or more code blocks (OB, FB, FC, or DB) in your program from unauthorized access. You create a password to limit access to the code block. The password-protection prevents unauthorized reading or modification of the code block. Without the password, you can read only the following information about the code block:

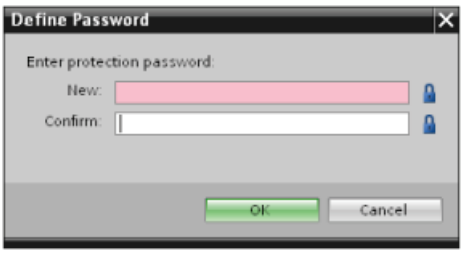
- Block title, block comment, and block properties
- Transfer parameters (IN, OUT, IN_OUT, Return)
- Call structure of the program
- Global tags in the cross references (without information on the point of use), but local tags are hidden

When you configure a block for "know-how" protection, the code within the block cannot be accessed except after entering the password.

Use the "Properties" task card of the code block to configure the know-how protection for that block. After opening the code block, select "Protection" from Properties.

1. In the Properties for the code block, click the "Protection" button to display the "Know-how protection" dialog.
2. Click the "Define" button to enter the password.

After entering and confirming the password, click "OK".



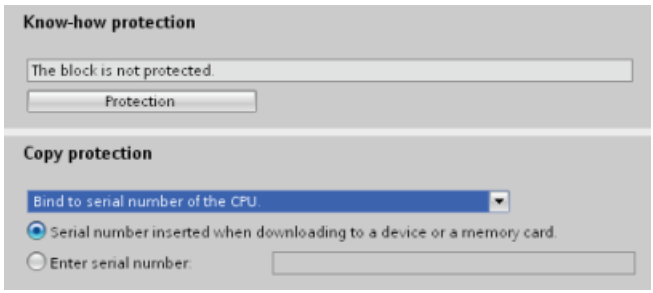
6.6.3 Copy protection

An additional security feature allows you to bind the program or code blocks for use with a specific memory card or CPU. This feature is especially useful for protecting your intellectual property. When you bind a program or block to a specific device, you restrict the program or code block for use only with a specific memory card or CPU. This feature allows you to distribute a program or code block electronically (such as over the Internet or through email) or by sending a memory cartridge.

Use the "Properties" task card of the code block to bind the block to a specific CPU or memory card.

1. After opening the code block, select "Protection".

2. From the drop-down list under "Copy protection" task, select the option to bind the code block either to a memory card or to a specific CPU.



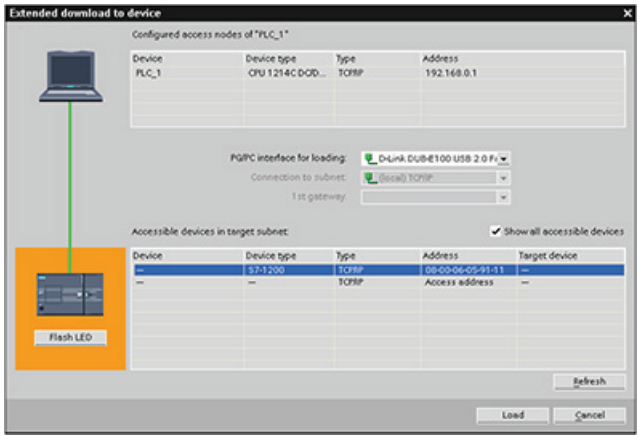
3. Select the type of copy protection and enter the serial number for the memory card or CPU.

Note

The serial number is case-sensitive.

6.7 Downloading the elements of your program

You can download the elements of your project from the programming device to the CPU. When you download a project, the CPU stores the user program (OBs, FCs, FBs and DBs) in permanent memory.



You can download your project from the programming device to your CPU from any of the following locations:

- "Project tree": Right-click the program element, and then click the context-sensitive "Download" selection.
- "Online" menu: Click the "Download to device" selection.
- Toolbar: Click the "Download to device" icon.

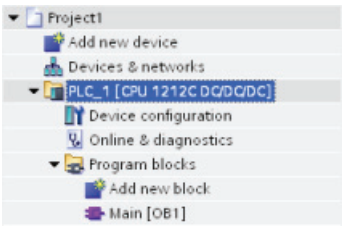
6.8 Uploading from the CPU

6.8.1 Copying elements of the project

You can also copy the program blocks from an online CPU or a memory card attached to your programming device.

Prepare the offline project for the copied program blocks:

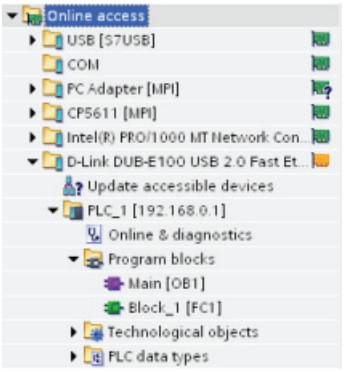
1. Add a CPU device that matches the online CPU.
2. Expand the CPU node once so that the "Program blocks" folder is visible.



To upload the program blocks from the online CPU to the offline project, follow these steps:

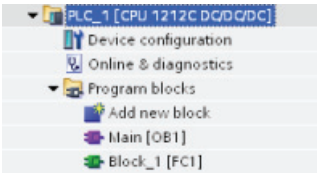
1. Click the "Program blocks" folder in the offline project.
2. Click the "Go online" button.
3. Click the "Upload" button.
4. Confirm your decision from the Upload dialog (Page 635).

As an alternative to the previous method, follow these steps:



1. From the project navigator, expand the node for "Online access" to select the program blocks in the online CPU:
2. Expand the node for the network, and double click "Update accessible devices".
3. Expand the node for the CPU.
4. Drag the "Program blocks" folder from the online CPU and drop the folder into the "Program blocks" folder of your offline project.
5. In the "Upload preview" dialog, select the box for "Continue", and then click the "Upload from device" button.

When the upload is complete, all of the program blocks, technology blocks, and tags will be displayed in the offline area.



Note

You can copy the program blocks from the online CPU to an existing program. The "Program-blocks" folder of the offline project does not have to be empty. However, the existing program will be deleted and replaced by the user program from the online CPU.

6.8.2 Using the compare function

You can use the "Compare" editor (Page 641) in STEP 7 to find differences between the online and offline projects. You might find this useful prior to uploading from the CPU.

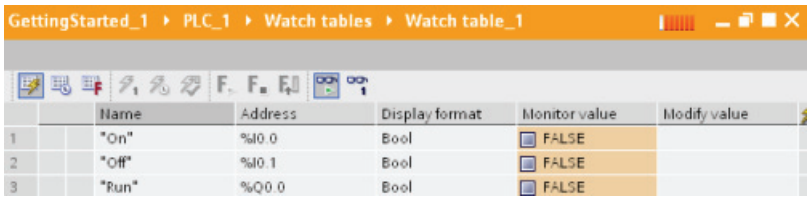
6.9 Debugging and testing the program

6.9.1 Monitor and modify data in the CPU

As shown in the following table, you can monitor and modify values in the online CPU.

Table 6- 5 Monitoring and modifying data with STEP 7

Editor	Monitor	Modify	Force
Watch table	Yes	Yes	No
Force table	Yes	No	Yes
Program editor	Yes	Yes	No
Tag table	Yes	No	No
DB editor	Yes	No	No



Monitoring with a watch table

Monitoring with the LAD editor

Refer to the "Online and diagnostics" chapter for more information about monitoring and modifying data in the CPU (Page 642).

6.9.2 Watch tables and force tables

You use "watch tables" for monitoring and modifying the values of a user program being executed by the online CPU. You can create and save different watch tables in your project to support a variety of test environments. This allows you to reproduce tests during commissioning or for service and maintenance purposes.

With a watch table, you can monitor and interact with the CPU as it executes the user program. You can display or change values not only for the tags of the code blocks and data blocks, but also for the memory areas of the CPU, including the inputs and outputs (I and Q), peripheral inputs (I:P), bit memory (M), and data blocks (DB).

With the watch table, you can enable the physical outputs (Q:P) of a CPU in STOP mode. For example, you can assign specific values to the outputs when testing the wiring for the CPU.

STEP 7 also provides a force table for "forcing" a tag to a specific value. For more information about forcing, see the section on forcing values in the CPU (Page 648) in the "Online and Diagnostics" chapter.

Note

The force values are stored in the CPU and not in the watch table.

You cannot force an input (or "I" address). However, you can force a peripheral input. To force a peripheral input, append a ":P" to the address (for example: "On:P").

6.9.3 Cross reference to show usage

The Inspector window displays cross-reference information about how a selected object is used throughout the complete project, such as the user program, the CPU and any HMI devices. The "Cross-reference" tab displays the instances where a selected object is being used and the other objects using it. The Inspector window also includes blocks which are only available online in the cross-references. To display the cross-references, select the "Show cross-references" command. (In the Project view, find the cross references in the "Tools" menu.)

Note

You do not have to close the editor to see the cross-reference information.

You can sort the entries in the cross-reference. The cross-reference list provides an overview of the use of memory addresses and tags within the user program.

- When creating and changing a program, you retain an overview of the operands, tags and block calls you have used.
- From the cross-references, you can jump directly to the point of use of operands and tags.
- During a program test or when troubleshooting, you are notified about which memory location is being processed by which command in which block, which tag is being used in which screen, and which block is called by which other block.

Table 6- 6 Elements of the cross reference

Column	Description
Object	Name of the object that uses the lower-level objects or that is being used by the lower-level objects
Quantity	Number of uses
Location	Each location of use, for example, network
Property	Special properties of referenced objects, for example, the tag names in multi-instance declarations
as	Shows additional information about the object, such as whether an instance DB is used as template or as a multiple instance

Column	Description
Access	Type of access, whether access to the operand is read access (R) and/or write access (W)
Address	Address of the operand
Type	Information on the type and language used to create the object
Path	Path of object in project tree

6.9.4 Call structure to examine the calling hierarchy

The call structure describes the call hierarchy of the block within your user program. It provides an overview of the blocks used, calls to other blocks, the relationships between blocks, the data requirements for each block, and the status of the blocks. You can open the program editor and edit blocks from the call structure.

Displaying the call structure provides you with a list of the blocks used in the user program. STEP 7 highlights the first level of the call structure and displays any blocks that are not called by any other block in the program. The first level of the call structure displays the OBs and any FCs, FBs, and DBs that are not called by an OB. If a code block calls another block, the called block is shown as an indentation under the calling block. The call structure only displays those blocks that are called by a code block.

You can selectively display only the blocks causing conflicts within the call structure. The following conditions cause conflicts:

- Blocks that execute any calls with older or newer code time stamps
- Blocks that call a block with modified interface
- Blocks that use a tag with modified address and/or data type
- Blocks that are called neither directly nor indirectly by an OB
- Blocks that call a non-existent or missing block

You can group several block calls and data blocks as a group. You use a drop-down list to see the links to the various call locations.

You can also perform a consistency check to show time stamp conflicts. Changing the time stamp of a block during or after the program is generated can lead to time stamp conflicts, which in turn cause inconsistencies among the blocks that are calling and being called.

- Most time stamp and interface conflicts can be corrected by recompiling the code blocks.
- If compilation fails to clear up inconsistencies, use the link in the "Details" column to go to the source of the problem in the program editor. You can then manually eliminate any inconsistencies.
- Any blocks marked in red must be recompiled.

Basic instructions 7

7.1 Bit logic

7.1.1 Bit logic contacts and coils

LAD and FBD are very effective for handling Boolean logic. While SCL is especially effective for complex mathematical computation and for project control structures, you can use SCL for Boolean logic.

LAD contacts

Table 7- 1 Normally open and normally closed contacts

LAD	SCL	Description

	`IF in THEN`
	`  Statement;`
	`ELSE`
	`  Statement;`
	`END_IF;`
		Normally open and normally closed contacts: You can connect contacts to other contacts and create your own combination logic. If the input bit you specify uses memory identifier I (input) or Q (output), then the bit value is read from the process-image register. The physical contact signals in your control process are wired to I terminals on the PLC. The CPU scans the wired input signals and continuously updates the corresponding state values in the process-image input register.
		You can specify an immediate read of a physical input using ":P" following the I offset (example: "%I3.4:P"). For an immediate read, the bit data values are read directly from the physical input instead of the process image. An immediate read does not update the process image.

	`IF NOT (in) THEN`
	`  Statement;`
	`ELSE`
	`  Statement;`
	`END_IF;`	

Table 7- 2 Data types for the parameters

Parameter	Data type	Description
IN	Bool	Assigned bit

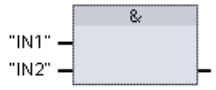
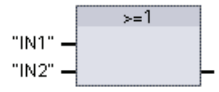
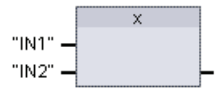
- The Normally Open contact is closed (ON) when the assigned bit value is equal to 1.
- The Normally Closed contact is closed (ON) when the assigned bit value is equal to 0.
- Contacts connected in series create AND logic networks.
- Contacts connected in parallel create OR logic networks.

FBD AND, OR, and XOR boxes

In FBD programming, LAD contact networks are transformed into AND (&), OR (>=1), and exclusive OR (x) box networks where you can specify bit values for the box inputs and outputs. You may also connect to other logic boxes and create your own logic combinations. After the box is placed in your network, you can drag the "Insert input" tool from the "Favorites" toolbar or instruction tree and then drop it onto the input side of the box to add more inputs. You can also right-click on the box input connector and select "Insert input".

Box inputs and outputs can be connected to another logic box, or you can enter a bit address or bit symbol name for an unconnected input. When the box instruction is executed, the current input states are applied to the binary box logic and, if true, the box output will be true.

Table 7- 3 AND, OR, and XOR boxes

FBD	SCL[1]	Description
	`out := in1 AND in2;`	All inputs of an AND box must be TRUE for the output to be TRUE.
	`out := in1 OR in2;`	Any input of an OR box must be TRUE for the output to be TRUE.
	`out := in1 XOR in2;`	An odd number of the inputs of an XOR box must be TRUE for the output to be TRUE.

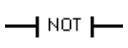
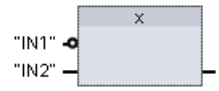
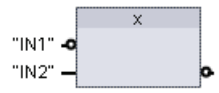
[1] For SCL: You must assign the result of the operation to a variable to be used for another statement.

Table 7- 4 Data types for the parameters

Parameter	Data type	Description
IN1, IN2	Bool	Input bit

NOT logic inverter

Table 7- 5 NOT Logic inverter

LAD	FBD	SCL	Description
	 	**NOT**	For FBD programming, you can drag the "Negate binary input" tool from the "Favorites" toolbar or instruction tree and then drop it on an input or output to create a logic inverter on that box connector. The LAD NOT contact inverts the logical state of power flow input. • If there is no power flow into the NOT contact, then there is power flow out. • If there is power flow into the NOT contact, then there is no power flow out.

Output coil and assignment box

The coil output instruction writes a value for an output bit. If the output bit you specify uses memory identifier Q, then the CPU turns the output bit in the process-image register on or off, setting the specified bit equal to power flow status. The output signals for your control actuators are wired to the Q terminals of the CPU. In RUN mode, the CPU system continuously scans your input signals, processes the input states according to your program logic, and then reacts by setting new output state values in the process-image output register. After each program execution cycle, the CPU system transfers the new output state reaction stored in the process-image register to the wired output terminals.

Table 7- 6 Output coil (LAD) and output assignment box (FBD)

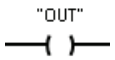
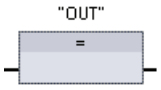
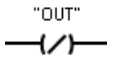
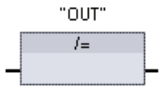
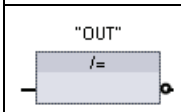
LAD	FBD	SCL	Description
		**out := <Boolean expression>;**	In FBD programming, LAD coils are transformed into assignment (= and /=) boxes where you specify a bit address for the box output. Box inputs and outputs can be connected to other box logic or you can enter a bit address. You can specify an immediate write of a physical output using ":P" following the Q offset (example: "%Q3.4:P"). For an immediate write, the bit data values are written to the process image output and directly to physical output.
	 	**out := NOT <Boolean expression>;**	

Table 7- 7 Data types for the parameters

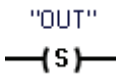
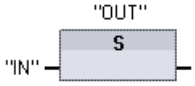
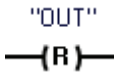
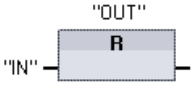
Parameter	Data type	Description
OUT	Bool	Assigned bit

- If there is power flow through an output coil or an FBD "=" box is enabled, then the output bit is set to 1.
- If there is no power flow through an output coil or an FBD "=" assignment box is not enabled, then the output bit is set to 0.
- If there is power flow through an inverted output coil or an FBD "/=" box is enabled, then the output bit is set to 0.
- If there is no power flow through an inverted output coil or an FBD "/=" box is not enabled, then the output bit is set to 1.

7.1.2 Set and reset instructions

Set and Reset 1 bit

Table 7- 8 S and R instructions

LAD	FBD	SCL	Description
		Not available	When S (Set) is activated, then the data value at the OUT address is set to 1. When S is not activated, OUT is not changed.
		Not available	When R (Reset) is activated, then the data value at the OUT address is set to 0. When R is not activated, OUT is not changed.

1 For LAD and FBD: These instructions can be placed anywhere in the network.

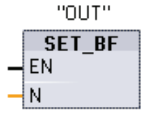
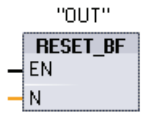
2 For SCL: You must write code to replicate this function within your application.

Table 7- 9 Data types for the parameters

Parameter	Data type	Description
IN (or connect to contact/gate logic)	Bool	Bit location to be monitored
OUT	Bool	Bit location to be set or reset

Set and Reset Bit Field

Table 7- 10 SET_BF and RESET_BF instructions

LAD[1]	FBD	SCL	Description
		Not available	When SET_BF is activated, a data value of 1 is assigned to "n" bits starting at address OUT. When SET_BF is not activated, OUT is not changed.
		Not available	RESET_BF writes a data value of 0 to "n" bits starting at address OUT. When RESET_BF is not activated, OUT is not changed.

1 For LAD and FBD: These instructions must be the right-most instruction in a branch.

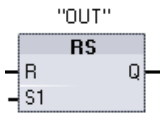
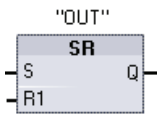
2 For SCL: You must write code to replicate this function within your application.

Table 7- 11 Data types for the parameters

Parameter	Data type	Description
OUT	Bool	Starting element of a bit field to be set or reset (Example: #MyArray[3])
n	Constant (UInt)	Number of bits to write

Set-dominant and Reset-dominant bit latches

Table 7- 12 RS and SR instructions

LAD / FBD	SCL	Description
	Not available	RS is a set dominant latch where the set dominates. If the set (S1) and reset (R) signals are both true, the output address OUT will be 1.
	Not available	SR is a reset dominant latch where the reset dominates. If the set (S) and reset (R1) signals are both true, the output address OUT will be 0.

1 For LAD and FBD: These instructions must be the right-most instruction in a branch.

2 For SCL: You must write code to replicate this function within your application.

Table 7- 13 Data types for the parameters

Parameter	Data type	Description
S, S1	Bool	Set input; 1 indicates dominance
R, R1	Bool	Reset input; 1 indicates dominance
OUT	Bool	Assigned bit output "OUT"
Q	Bool	Follows state of "OUT" bit

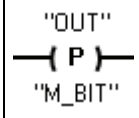
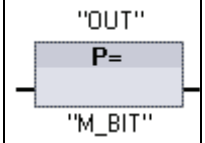
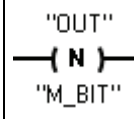
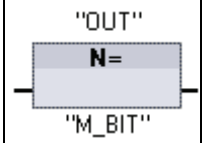
The OUT parameter specifies the bit address that is set or reset. The optional OUT output Q reflects the signal state of the "OUT" address.

Instruction	S1	R	"OUT" bit
RS	0	0	Previous state
	0	1	0
	1	0	1
	1	1	1
	S	**R1**	
SR	0	0	Previous state
	0	1	0
	1	0	1
	1	1	0

7.1.3 Positive and negative edge instructions

Table 7- 14 Positive and negative transition detection

LAD	FBD	SCL	Description
"IN" P "M_BIT"	"IN" P "M_BIT"	Not available	LAD: The state of this contact is TRUE when a positive transition (OFF-to-ON) is detected on the assigned "IN" bit. The contact logic state is then combined with the power flow in state to set the power flow out state. The P contact can be located anywhere in the network except the end of a branch. FBD: The output logic state is TRUE when a positive transition (OFF-to-ON) is detected on the assigned input bit. The P box can only be located at the beginning of a branch.
"IN" N "M_BIT"	"IN" N "M_BIT"	Not available	LAD: The state of this contact is TRUE when a negative transition (ON-to-OFF) is detected on the assigned input bit. The contact logic state is then combined with the power flow in state to set the power flow out state. The N contact can be located anywhere in the network except the end of a branch. FBD: The output logic state is TRUE when a negative transition (ON-to-OFF) is detected on the assigned input bit. The N box can only be located at the beginning of a branch.

LAD	FBD	SCL	Description
		Not available	LAD: The assigned bit "OUT" is TRUE when a positive transition (OFF-to-ON) is detected on the power flow entering the coil. The power flow in state always passes through the coil as the power flow out state. The P coil can be located anywhere in the network. FBD: The assigned bit "OUT" is TRUE when a positive transition (OFF-to-ON) is detected on the logic state at the box input connection or on the input bit assignment if the box is located at the start of a branch. The input logic state always passes through the box as the output logic state. The P= box can be located anywhere in the branch.
		Not available	LAD: The assigned bit "OUT" is TRUE when a negative transition (ON-to-OFF) is detected on the power flow entering the coil. The power flow in state always passes through the coil as the power flow out state. The N coil can be located anywhere in the network. FBD: The assigned bit "OUT" is TRUE when a negative transition (ON-to-OFF) is detected on the logic state at the box input connection or on the input bit assignment if the box is located at the start of a branch. The input logic state always passes through the box as the output logic state. The N= box can be located anywhere in the branch.

[1] For SCL: You must write code to replicate this function within your application.

Table 7- 15 P_TRIG and N_TRIG instructions

LAD / FBD	SCL	Description
	Not available	The Q output power flow or logic state is TRUE when a positive transition (OFF-to-ON) is detected on the CLK input state (FBD) or CLK power flow in (LAD). In LAD, the P_TRIG instruction cannot be located at the beginning or end of a network. In FBD, the P_TRIG instruction can be located anywhere except the end of a branch.
	Not available	The Q output power flow or logic state is TRUE when a negative transition (ON-to-OFF) is detected on the CLK input state (FBD) or CLK power flow in (LAD). In LAD, the N_TRIG instruction cannot be located at the beginning or end of a network. In FBD, the N_TRIG instruction can be located anywhere except the end of a branch.

[1] For SCL: You must write code to replicate this function within your application.

Table 7- 16 Data types for the parameters (P and N contacts/coils, P=, N=, P_TRIG and N_TRIG)

Parameter	Data type	Description
M_BIT	Bool	Memory bit in which the previous state of the input is saved
IN	Bool	Input bit whose transition edge is to be detected
OUT	Bool	Output bit which indicates a transition edge was detected
CLK	Bool	Power flow or input bit whose transition edge is to be detected
Q	Bool	Output which indicates an edge was detected

All edge instructions use a memory bit (M_BIT) to store the previous state of the input signal being monitored. An edge is detected by comparing the state of the input with the state of the memory bit. If the states indicate a change of the input in the direction of interest, then an edge is reported by writing the output TRUE. Otherwise, the output is written FALSE.

Note

Edge instructions evaluate the input and memory-bit values each time they are executed, including the first execution. You must account for the initial states of the input and memory bit in your program design either to allow or to avoid edge detection on the first scan.

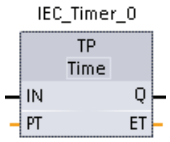
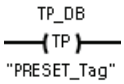
Because the memory bit must be maintained from one execution to the next, you should use a unique bit for each edge instruction, and you should not use this bit any other place in your program. You should also avoid temporary memory and memory that can be affected by other system functions, such as an I/O update. Use only M, global DB, or Static memory (in an instance DB) for M_BIT memory assignments.

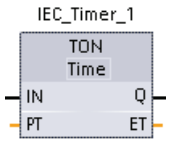
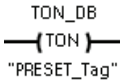
7.2 Timers

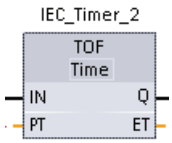
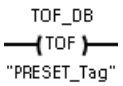
You use the timer instructions to create programmed time delays. The number of timers that you can use in your user program is limited only by the amount of memory in the CPU. Each timer uses a 16 byte IEC_Timer data type DB structure to store timer data that is specified at the top of the box or coil instruction. STEP 7 automatically creates the DB when you insert the instruction.

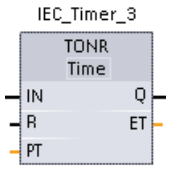
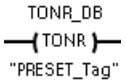
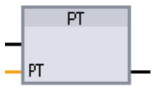
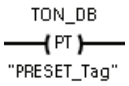
Table 7- 17 Timer instructions

LAD / FBD boxes	LAD coils	SCL	Description

		`"IEC_Timer_0_DB".TP(`
		`IN:=_bool_in_,`
		`PT:=_time_in_,`
		`Q=>_bool_out_,`
		`ET=>_time_out_);`
			The TP timer generates a pulse with a preset width time.

		`"IEC_Timer_0_DB".TON (`
		`IN:=_bool_in_,`
		`PT:=_time_in_,`
		`Q=>_bool_out_,`
		`ET=>_time_out_);`
			The TON timer sets output Q to ON after a preset time delay.

		`"IEC_Timer_0_DB".TOF (`
		`IN:=_bool_in_,`
		`PT:=_time_in_,`
		`Q=>_bool_out_,`
		`ET=>_time_out_);`
			The TOF timer resets output Q to OFF after a preset time delay.

LAD / FBD boxes	LAD coils	SCL	Description
		`"IEC_Timer_0_DB".TONR ( IN:=_bool_in_, R:=_bool_in_ PT:=_time_in_, Q=>_bool_out_, ET=>_time_out_);`	The TONR timer sets output Q to ON after a preset time delay. Elapsed time is accumulated over multiple timing periods until the R input is used to reset the elapsed time.
FBD only: 		(No SCL equivalent)	The PT (Preset timer) coil loads a new PRESET time value in the specified IEC_Timer.
FBD only: 		(No SCL equivalent)	The RT (Reset timer) coil resets the specified IEC_Timer.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL examples, "IEC_Timer_0_DB" is the name of the instance DB.

Table 7- 18 Data types for the parameters

Parameter	Data type	Description
Box: IN Coil: Power flow	Bool	TP, TON, and TONR: Box: 0=Disable timer, 1=Enable timer Coil: No power flow=Disable timer, Power flow=Enable timer TOF: Box: 0=Enable timer, 1=Disable timer Coil: No power flow=Enable timer, Power flow=Disable timer
R	Bool	TONR box only: 0=No reset 1= Reset elapsed time and Q bit to 0
Box: PT Coil: "PRESET_Tag"	Time	Timer box or coil: Preset time input
Box: Q Coil: DBdata.Q	Bool	Timer box: Q box output or Q bit in the timer DB data Timer coil: you can only address the Q bit in the timer DB data
Box: ET Coil: DBdata.ET	Time	Timer box: ET (elapsed time) box output or ET time value in the timer DB data Timer coil: you can only address the ET time value in the timer DB data.

Table 7- 19 Effect of value changes in the PT and IN parameters

Timer	Changes in the PT and IN box parameters and the corresponding coil parameters
TP	• Changing PT has no effect while the timer runs. • Changing IN has no effect while the timer runs.
TON	• Changing PT has no effect while the timer runs. • Changing IN to FALSE, while the timer runs, resets and stops the timer.
TOF	• Changing PT has no effect while the timer runs. • Changing IN to TRUE, while the timer runs, resets and stops the timer.
TONR	• Changing PT has no effect while the timer runs, but has an effect when the timer resumes. • Changing IN to FALSE, while the timer runs, stops the timer but does not reset the timer. Changing IN back to TRUE will cause the timer to start timing from the accumulated time value.

PT (preset time) and ET (elapsed time) values are stored in the specified IEC_TIMER DB data as signed double integers that represent milliseconds of time. TIME data uses the T# identifier and can be entered as a simple time unit (T#200ms or 200) and as compound time units like T#2s_200ms.

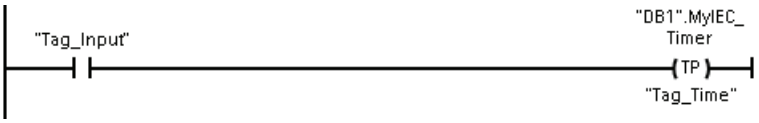
Table 7- 20 Size and range of the TIME data type

Data type	Size	Valid number ranges[1]
TIME	32 bits, stored as DInt data	T#-24d_20h_31m_23s_648ms to T#24d_20h_31m_23s_647ms Stored as -2,147,483,648 ms to +2,147,483,647 ms

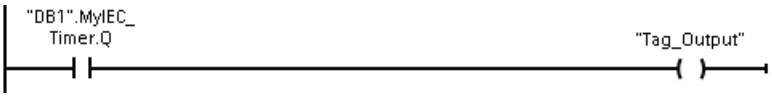
[1] The negative range of the TIME data type shown above cannot be used with the timer instructions. Negative PT (preset time) values are set to zero when the timer instruction is executed. ET (elapsed time) is always a positive value.

Timer coil example

The -(TP)-, -(TON)-, -(TOF)-, and -(TONR)- timer coils must be the last instruction in a LAD network. As shown in the timer example, a contact instruction in a subsequent network evaluates the Q bit in a timer coil's IEC_Timer DB data. Likewise, you must address the ELAPSED element in the IEC_timer DB data if you want to use the elapsed time value in your program.



The pulse timer is started on a 0 to 1 transition of the Tag_Input bit value. The timer runs for the time specified by Tag_Time time value.



As long as the timer runs, the state of DB1.MyIEC_Timer.Q=1 and the Tag_Output value=1. When the Tag_Time value has elapsed, then DB1.MyIEC_Timer.Q=0 and the Tag_Output value=0.

Reset timer -(RT)- and Preset timer -(PT)- coils

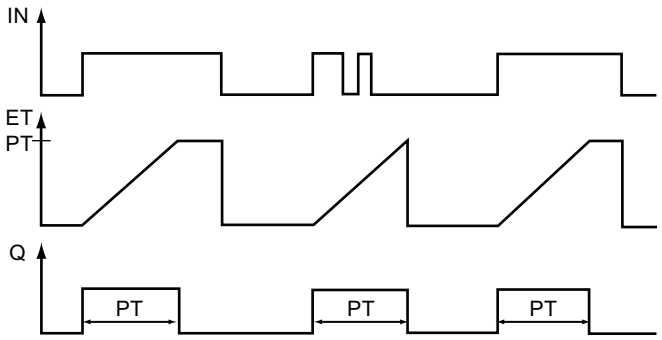
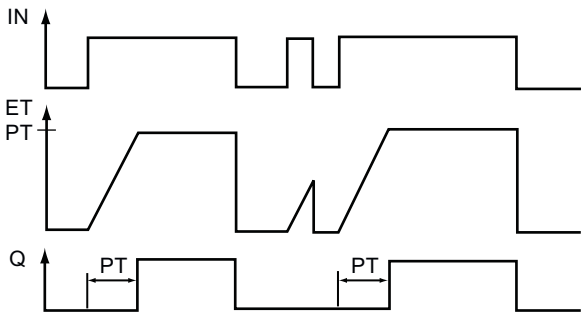
These coil instructions can be used with box or coil timers and can be placed in a mid-line position. The coil output power flow status is always the same as the coil input status. When the -(RT)- coil is activated, the ELAPSED time element of the specified IEC_Timer DB data is reset to 0. When the -(PT)- coil is activated, the PRESET time element of the specified IEC_Timer DB data is reset to 0.

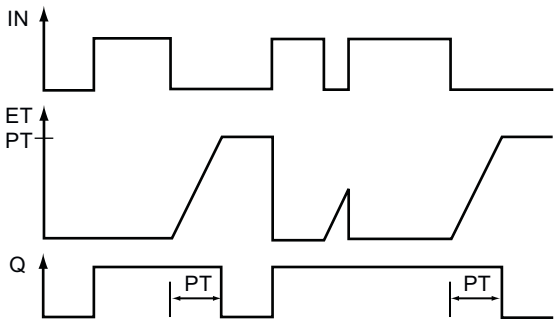
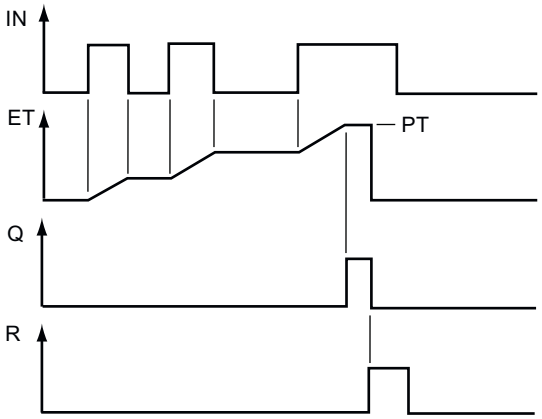
Note

When you place timer instructions in an FB, you can select the "Multi-instance data block" option. The timer structure names can be different with separate data structures, but the timer data is contained in a single data block and does not require a separate data block for each timer. This reduces the processing time and data storage necessary for handling the timers. There is no interaction between the timer data structures in the shared multi-instance DB.

Operation of the timers

Table 7- 21 Types of IEC timers

Timer	Timing diagram
TP: Pulse timer The TP timer generates a pulse with a preset width time.	
TON: ON-delay timer The TON timer sets output Q to ON after a preset time delay.	

Timer	Timing diagram
TOF: OFF-delay timer The TOF timer resets output Q to OFF after a preset time delay.	
TONR: ON-delay Retentive timer The TONR timer sets output Q to ON after a preset time delay. Elapsed time is accumulated over multiple timing periods until the R input is used to reset the elapsed time.	

Note

In the CPU, no dedicated resource is allocated to any specific timer instruction. Instead, each timer utilizes its own timer structure in DB memory and a continuously-running internal CPU timer to perform timing.

When a timer is started due to an edge change on the input of a TP, TON, TOF, or TONR instruction, the value of the continuously-running internal CPU timer is copied into the START member of the DB structure allocated for this timer instruction. This start value remains unchanged while the timer continues to run, and is used later each time the timer is updated. Each time the timer is started, a new start value is loaded into the timer structure from the internal CPU timer.

When a timer is updated, the start value described above is subtracted from the current value of the internal CPU timer to determine the elapsed time. The elapsed time is then compared with the preset to determine the state of the timer Q bit. The ELAPSED and Q members are then updated in the DB structure allocated for this timer. Note that the elapsed time is clamped at the preset value (the timer does not continue to accumulate elapsed time after the preset is reached).

A timer update is performed when and only when:

- A timer instruction (TP, TON, TOF, or TONR) is executed
- The "ELAPSED" member of the timer structure in DB is referenced directly by an instruction
- The "Q" member of the timer structure in DB is referenced directly by an instruction

Timer programming

The following consequences of timer operation should be considered when planning and creating your user program:

- You can have multiple updates of a timer in the same scan. The timer is updated each time the timer instruction (TP, TON, TOF, TONR) is executed and each time the ELAPSED or Q member of the timer structure is used as a parameter of another executed instruction. This is an advantage if you want the latest time data (essentially an immediate read of the timer). However, if you desire to have consistent values throughout a program scan, then place your timer instruction prior to all other instructions that need these values, and use tags from the Q and ET outputs of the timer instruction instead of the ELAPSED and Q members of the timer DB structure.
- You can have scans during which no update of a timer occurs. It is possible to start your timer in a function, and then cease to call that function again for one or more scans. If no other instructions are executed which reference the ELAPSED or Q members of the timer structure, then the timer will not be updated. A new update will not occur until either the timer instruction is executed again or some other instruction is executed using ELAPSED or Q from the timer structure as a parameter.
- Although not typical, you can assign the same DB timer structure to multiple timer instructions. In general, to avoid unexpected interaction, you should only use one timer instruction (TP, TON, TOF, TONR) per DB timer structure.
- Self-resetting timers are useful to trigger actions that need to occur periodically. Typically, self-resetting timers are created by placing a normally-closed contact which references the timer bit in front of the timer instruction. This timer network is typically located above one or more dependent networks that use the timer bit to trigger actions. When the timer expires (elapsed time reaches preset value), the timer bit is ON for one scan, allowing the dependent network logic controlled by the timer bit to execute. Upon the next execution of the timer network, the normally closed contact is OFF, thus resetting the timer and clearing the timer bit. The next scan, the normally closed contact is ON, thus restarting the timer. When creating self-resetting timers such as this, do not use the "Q" member of the timer DB structure as the parameter for the normally-closed contact in front of the timer instruction. Instead, use the tag connected to the "Q" output of the timer instruction for this purpose. The reason to avoid accessing the Q member of the timer DB structure is because this causes an update to the timer and if the timer is updated due to the normally closed contact, then the contact will reset the timer instruction immediately. The Q output of the timer instruction will not be ON for the one scan and the dependent networks will not execute.

Time data retention after a RUN-STOP-RUN transition or a CPU power cycle

If a run mode session is ended with stop mode or a CPU power cycle and a new run mode session is started, then the timer data stored in the previous run mode session is lost, unless the timer data structure is specified as retentive (TP, TON, TOF, and TONR timers).

When you accept the defaults in the call options dialog after you place a timer instruction in the program editor, you are automatically assigned an instance DB which **cannot be made retentive**. To make your timer data retentive, you must either use a global DB or a Multi-instance DB.

Assign a global DB to store timer data as retentive data

This option works regardless of where the timer is placed (OB, FC, or FB).

1. Create a global DB:
   - Double-click "Add new block" from the Project tree
   - Click the data block (DB) icon
   - For the Type, choose global DB
   - If you want to be able to select individual data elements in this DB as retentive, be sure the DB type "Optimized" box is checked. The other DB type option "Standard - compatible with S7-300/400" only allows setting all DB data elements retentive or none retentive.
   - Click OK
2. Add timer structure(s) to the DB:
   - In the new global DB, add a new static tag using data type IEC_Timer.
   - In the "Retain" column, check the box so that this structure will be retentive.
   - Repeat this process to create structures for all the timers that you want to store in this DB. You can either place each timer structure in a unique global DB, or you can place multiple timer structures into the same global DB. You can also place other static tags besides timers in this global DB. Placing multiple timer structures into the same global DB allows you to reduce your overall number of blocks.
   - Rename the timer structures if desired.
3. Open the program block for editing where you want to place a retentive timer (OB, FC, or FB).
4. Place the timer instruction at the desired location.
5. When the call options dialog appears, click the cancel button.
6. On the top of the new timer instruction, type the name (do not use the helper to browse) of the global DB and timer structure that you created above (example: "Data_block_3.Static_1").

Assign a multi-instance DB to store timer data as retentive data

This option only works if you place the timer in an FB.

This option depends upon whether the FB was created with "Optimized" block access (allows symbolic access only). Once the FB has been created, you cannot change the checkbox for "Optimized"; it must be chosen correctly when the FB is created, on the first screen after selecting "Add new block" from the tree. To verify how the access attribute is configured for an existing FB, right-click on the FB in the Project tree, choose properties, and then choose attributes.

If the FB was created with the "Optimized" box checked (allows symbolic access only):

1. Open the FB for edit.
2. Place the timer instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the Multi instance icon. The Multi Instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the timer if desired.
5. Click OK. The timer instruction appears in the editor, and the IEC_TIMER structure appears in the FB Interface under Static.
6. If necessary, open the FB interface editor (may have to click on the small arrow to expand the view).
7. Under Static, locate the timer structure that was just created for you.
8. In the Retain column for this timer structure, change the selection to "Retain". Whenever this FB is called later from another program block, an instance DB will be created with this interface definition which contains the timer structure marked as retentive.

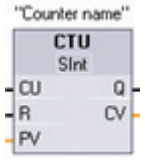
If the FB was created with the "Standard - compatible with S7-300/400" box checked (allows symbolic and direct access):

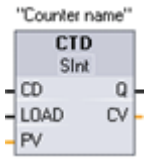
1. Open the FB for edit.
2. Place the timer instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the multi instance icon. The multi instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the timer if desired.
5. Click OK. The timer instruction appears in the editor, and the IEC_TIMER structure appears in the FB Interface under Static.
6. Open the block that will use this FB.
7. Place this FB at the desired location. Doing so results in the creation of an instance data block for this FB.
8. Open the instance data block created when you placed the FB in the editor.
9. Under Static, locate the timer structure of interest. In the Retain column for this timer structure, check the box to make this structure retentive.

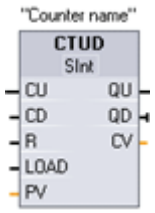
7.3 Counters

Table 7- 22 Counter instructions

LAD / FBD	SCL	Description

	`"IEC_Counter_0_DB".CTU(`
	`CU:=_bool_in,`
	`R:=_bool_in,`
	`PV:=_int_in,`
	`Q=>_bool_out,`
	`CV=>_int_out);`
		Use the counter instructions to count internal program events and external process events. Each counter uses a structure stored in a data block to maintain counter data. You assign the data block when the counter instruction is placed in the editor.
		• CTU is a count-up counter
		• CTD is a count-down counter
		• CTUD is a count-up-and-down counter

	`"IEC_Counter_0_DB".CTD(`
	`CD:=_bool_in,`
	`LD:=_bool_in,`
	`PV:=_int_in,`
	`Q=>_bool_out,`
	`CV=>_int_out);`	

	`"IEC_Counter_0_DB".CTUD(`
	`CU:=_bool_in,`
	`CD:=_bool_in,`
	`R:=_bool_in,`
	`LD:=_bool_in,`
	`PV:=_int_in,`
	`QU=>_bool_out,`
	`QD=>_bool_out,`
	`CV=>_int_out);`	

1 For LAD and FBD: Select the count value data type from the drop-down list below the instruction name.

2 STEP 7 automatically creates the DB when you insert the instruction.

3 In the SCL examples, "IEC_Counter_0_DB" is the name of the instance DB.

Table 7- 23 Data types for the parameters

Parameter	Data type[1]	Description
CU, CD	Bool	Count up or count down, by one count
R (CTU, CTUD)	Bool	Reset count value to zero
LD (CTD, CTUD)	Bool	Load control for preset value
PV	SInt, Int, DInt, USInt, UInt, UDInt	Preset count value
Q, QU	Bool	True if CV >= PV
QD	Bool	True if CV <= 0
CV	SInt, Int, DInt, USInt, UInt, UDInt	Current count value

1 The numerical range of count values depends on the data type you select. If the count value is an unsigned integer type, you can count down to zero or count up to the range limit. If the count value is a signed integer, you can count down to the negative integer limit and count up to the positive integer limit.

The number of counters that you can use in your user program is limited only by the amount of memory in the CPU. Counters use the following amount of memory:

- For SInt or USInt data types, the counter instruction uses 3 bytes.
- For Int or UInt data types, the counter instruction uses 6 bytes.
- For DInt or UDInt data types, the counter instruction uses 12 bytes.

These instructions use software counters whose maximum counting rate is limited by the execution rate of the OB in which they are placed. The OB that the instructions are placed in must be executed often enough to detect all transitions of the CU or CD inputs. For faster counting operations, see the CTRL_HSC instruction (Page 315).

Note

When you place counter instructions in an FB, you can select the multi-instance DB option, the counter structure names can be different with separate data structures, but the counter data is contained in a single DB and does not require a separate DB for each counter. This reduces the processing time and data storage necessary for the counters. There is no interaction between the counter data structures in the shared multi-instance DB.

Operation of the counters

Table 7- 24 Operation of the CTU counter

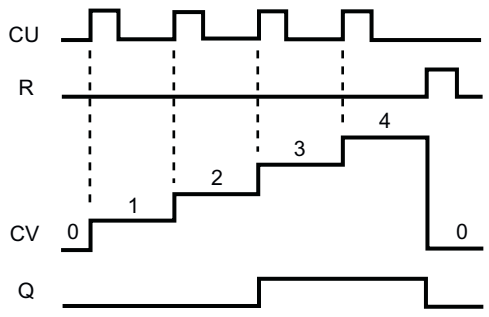
Counter	Operation
The CTU counter counts up by 1 when the value of parameter CU changes from 0 to 1. The CTU timing diagram shows the operation for an unsigned integer count value (where PV = 3). • If the value of parameter CV (current count value) is greater than or equal to the value of parameter PV (preset count value), then the counter output parameter Q = 1. • If the value of the reset parameter R changes from 0 to 1, then the current count value is reset to 0.	

Table 7- 25 Operation of the CTD counter

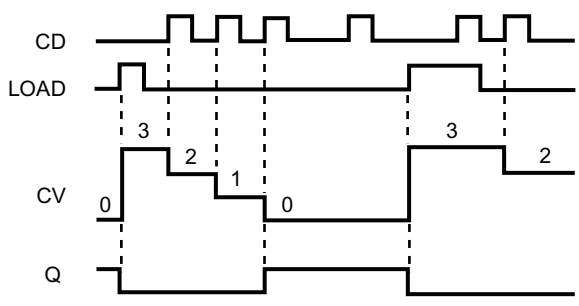
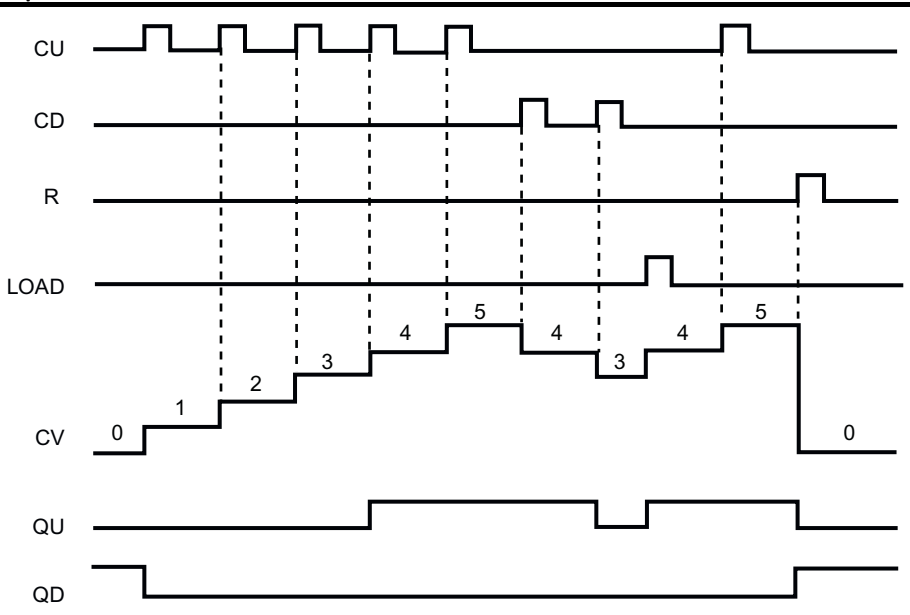
Counter	Operation
The CTD counter counts down by 1 when the value of parameter CD changes from 0 to 1. The CTD timing diagram shows the operation for an unsigned integer count value (where PV = 3). • If the value of parameter CV (current count value) is equal to or less than 0, the counter output parameter Q = 1. • If the value of parameter LOAD changes from 0 to 1, the value at parameter PV (preset value) is loaded to the counter as the new CV (current count value).	

Table 7- 26 Operation of the CTUD counter

Counter	Operation
The CTUD counter counts up or down by 1 on the 0 to 1 transition of the count up (CU) or count down (CD) inputs. The CTUD timing diagram shows the operation for an unsigned integer count value (where PV = 4). • If the value of parameter CV is equal to or greater than the value of parameter PV, then the counter output parameter QU = 1. • If the value of parameter CV is less than or equal to zero, then the counter output parameter QD = 1. • If the value of parameter LOAD changes from 0 to 1, then the value at parameter PV is loaded to the counter as the new CV. • If the value of the reset parameter R is changes from 0 to 1, the current count value is reset to 0.	

Counter data retention after a RUN-STOP-RUN transition or a CPU power cycle

If a run mode session is ended with stop mode or a CPU power cycle and a new run mode session is started, then the counter data stored in the previous run mode session is lost, unless the counter data structure is specified as retentive (CTU, CTD, and CTUD counters).

When you accept the defaults in the call options dialog after you place a counter instruction in the program editor, you are automatically assigned an instance DB which **cannot be made retentive**. To make your counter data retentive, you must either use a global DB or a Multi-instance DB.

Assign a global DB to store timer data as retentive data

This option works regardless of where the counter is placed (OB, FC, or FB).

1. Create a global DB:
   - Double-click "Add new block" from the Project tree
   - Click the data block (DB) icon
   - For the Type, choose global DB
   - If you want to be able to select individual items in this DB as retentive, be sure the symbolic-access-only box is checked.
   - Click OK
2. Add counter structure(s) to the DB:
   - In the new global DB, add a new static tag using one of the counter data types. Be sure to consider the Type you want to use for your Preset and Count values.

Counter Data Type	Corresponding Type for the Preset and Count Values
IEC_Counter	INT
IEC_SCounter	SINT
IEC_DCounter	DINT
IEC_UCounter	UINT
IEC_USCounter	USINT
IEC_UDCounter	UDINT

1. In the "Retain" column, check the box so that this structure will be retentive.
   - Repeat this process to create structures for all the counters that you want to store in this DB. You can either place each counter structure in a unique global DB, or you can place multiple counter structures into the same global DB. You can also place other static tags besides counters in this global DB. Placing multiple counter structures into the same global DB allows you to reduce your overall number of blocks.
   - Rename the counter structures if desired.
2. Open the program block for editing where you want to place a retentive counter (OB, FC, or FB).
3. Place the counter instruction at the desired location.
4. When the call options dialog appears, click the cancel button. You should now see a new counter instruction which has "???" both just above and just below the instruction name.
5. On the top of the new counter instruction, type the name (do not use the helper to browse) of the global DB and counter structure that you created above (example: "Data_block_3.Static_1"). This causes the corresponding preset and count value type to be filled in (example: UInt for an IEC_UCounter structure).

Assign a multi-instance DB to store counter data as retentive data

This option only works if you place the counter in an FB.

This option depends upon whether the FB was created as symbolic access only. Once the FB has been created, you cannot change the checkbox for "Symbolic access only"; it must be chosen correctly when the FB is created, on the first screen after selecting "Add new block" from the tree. To see how this box is configured for an existing FB, right-click on the FB in the Project tree, choose properties, and then choose attributes.

If the FB was created with the "Symbolic access only" box checked:

1. Open the FB for edit.
2. Place the counter instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the Multi instance icon. The Multi Instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the counter if desired.
5. Click OK. The counter instruction appears in the editor with type INT for the preset and count values, and the IEC_COUNTER structure appears in the FB Interface under Static.
6. If desired, change the type in the counter instruction from INT to one of the other types. The counter structure will change correspondingly.

Type shown in counter instruction (for preset and count values)	Corresponding structure Type shown in FB interface
INT	IEC_Counter
SINT	IEC_SCounter
DINT	IEC_DCounter
UINT	IEC_UCounter
USINT	IEC_USCounter
UDINT	IEC_UDCounter

1. If necessary, open the FB interface editor (may have to click on the small arrow to expand the view).
2. Under Static, locate the counter structure that was just created for you.
3. In the Retain column for this counter structure, change the selection to "Retain". Whenever this FB is called later from another program block, an instance DB will be created with this interface definition which contains the counter structure marked as retentive.

If the FB was created with the "Symbolic access only" box *not* checked:

1. Open the FB for edit.
2. Place the counter instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the multi instance icon. The multi instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the counter if desired.
5. Click OK. The counter instruction appears in the editor with type INT for the preset and count value, and the IEC_COUNTER structure appears in the FB Interface under Static.
6. If desired, change the type in the counter instruction from INT to one of the other types. The counter structure will change correspondingly.

Type shown in counter instruction (for preset and count values)	Corresponding structure Type shown in FB interface
INT	IEC_Counter
SINT	IEC_SCounter
DINT	IEC_DCounter
UINT	IEC_UCounter
USINT	IEC_USCounter
UDINT	IEC_UDCounter

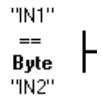
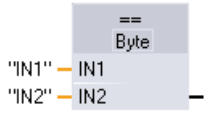
1. Open the block that will use this FB.
2. Place this FB at the desired location. Doing so results in the creation of an instance data block for this FB.
3. Open the instance data block created when you placed the FB in the editor.
4. Under Static, locate the counter structure of interest. In the Retain column for this counter structure, check the box to make this structure retentive.

7.4 Compare

7.4.1 Compare

Table 7- 27 Compare instructions

LAD	FBD	SCL	Description

		`out := in1 == in2;`
		or
		`IF in1== in2,`
		`THEN out := 1;`
		`ELSE out := 0;`
		`END_IF;`
			Compares two values of the same data type. When the LAD contact comparison is TRUE, then the contact is activated. When the FBD box comparison is TRUE, then the box output is TRUE.

[1] For LAD and FBD: Click the instruction name (such as "==") to change the comparison type from the drop-down list. Click the "???" and select data type from the drop-down list.

Table 7- 28 Data types for the parameters

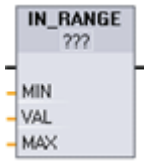
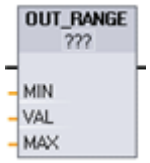
Parameter	Data type	Description
IN1, IN2	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, String, Char, Time, DTL, Constant	Values to compare

Table 7- 29 Comparison descriptions

Relation type	The comparison is true if ...
==	IN1 is equal to IN2
<>	IN1 is not equal to IN2
>=	IN1 is greater than or equal to IN2
<=	IN1 is less than or equal to IN2
>	IN1 is greater than IN2
<	IN1 is less than IN2

7.4.2 In-range and Out-of-range instructions

Table 7- 30 In Range and Out of Range instructions

LAD / FBD	SCL	Description
	`out := IN_RANGE(min, val, max);`	Tests whether an input value is in or out of a specified value range. If the comparison is TRUE, then the box output is TRUE.
	`out := OUT_RANGE(min, val, max);`	

[1] For LAD and FBD: Click the "???" and select the data type from the drop-down list.

Table 7- 31 Data types for the parameters

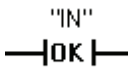
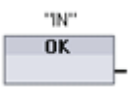
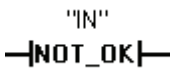
Parameter	Data type[1]	Description
MIN, VAL, MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Comparator inputs

[1] The input parameters MIN, VAL, and MAX must be the same data type.

- The IN_RANGE comparison is true if: MIN <= VAL <= MAX
- The OUT_RANGE comparison is true if: VAL < MIN or VAL > MAX

7.4.3 OK and Not OK instructions

Table 7- 32 OK and Not OK instructions

LAD	FBD	SCL	Description
		Not available	Tests whether an input data reference is a valid real number according to IEEE specification 754. For SCL, OK allows you to check the errors in the execution of a statement. OK is a predefined local tag that stores a Bool value. You can use the NOT keyword in conjunction with OK to evaluate the operation.
		Not available	

1 For LAD and FBD: When the LAD contact is TRUE, the contact is activated and passes power flow. When the FBD box is TRUE, then the box output is TRUE.

Table 7- 33 Data types for the parameter

Parameter	Data type	Description
IN	Real, LReal	Input data

Table 7- 34 Operation

Instruction	The Real number test is TRUE if:
OK	The input value is a valid real number [1]
NOT_OK	The input value is not a valid real number [1]

1 A Real or LReal value is invalid if it is +/- INF (infinity), NaN (Not a Number), or if it is a denormalized value. A denormalized value is a number very close to zero. The CPU substitutes a zero for a denormalized value in calculations.

When the CPU starts to execute an SCL code block, the CPU sets OK to TRUE. An error that occurs during the execution of an operation (for example, division by zero) sets OK to FALSE. During the execution of the SCL code, statements can query the OK parameter or can set OK to either TRUE or FALSE.

Table 7- 35 Using OK for checking the operation of an operation

```
SCL                                  Comment
OK := TRUE;                          // Set OK to TRUE
Division:= 1 / "IN";                 // Division operation
IF OK THEN                           // Check for valid operation (such
                                     as IN <>0).
...                                  // Statements for valid operation.
ELSE                                 // Invalid operation (such as IN =
                                     0).
...                                  // Statements handling invalid
                                     operation
END_IF;
```

You can configure the SCL compiler to write the value of OK to the output parameter ENO after the execution of the code block finishes. Refer to the section on EN and ENO (Page 152).

7.5 Math

7.5.1 Calculate instruction

Table 7- 36 CALCULATE instruction

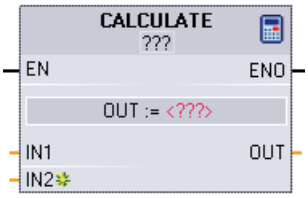
LAD / FBD	SCL	Description
	Use the standard SCL math expressions to create the equation.	The CALCULATE instruction lets you create a math function that operates on inputs (IN1, IN2, .. INn) and produces the result at OUT, according to the equation that you define. • Select a data type first. All inputs and the output must be the same data type. • To add another input, click the icon at the last input.

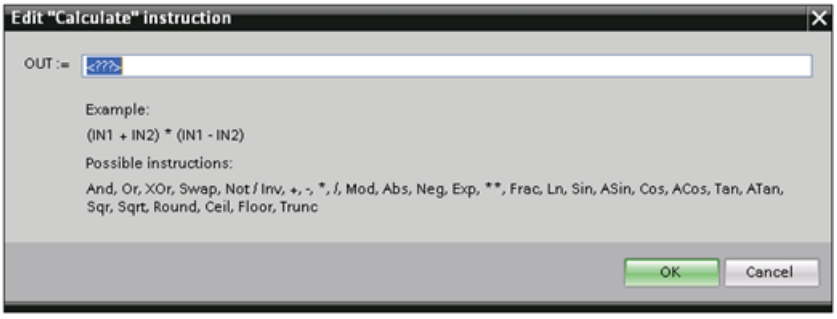
Table 7- 37 Data types for the parameters

Parameter	Data type[1]
IN1, IN2, ..INn	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord

[1] The IN and OUT parameters must be the same data type (with implicit conversions of the input parameters). For example: A SINT value for an input would be converted to an INT or a REAL value if OUT is an INT or REAL

Click the calculator icon to open the dialog and define your math function. You enter your equation as inputs (such as IN1 and IN2) and operations. When you click "OK" to save the function, the dialog automatically creates the inputs for the CALCULATE instruction.

An example and a list of possible math operations you can include is shown at the bottom of the editor.



Note

You also must create an input for any constants in your function. The constant value would then be entered in the associated input for the CALCULATE instruction.

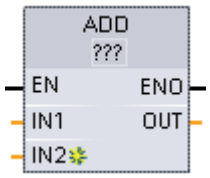
By entering constants as inputs, you can copy the CALCULATE instruction to other locations in your user program without having to change the function. You then can change the values or tags of the inputs for the instruction without modifying the function.

When CALCULATE is executed and all the individual operations in the calculation complete successfully, then the ENO = 1. Otherwise, ENO = 0.

7.5.2 Add, subtract, multiply and divide instructions

Table 7- 38 Add, subtract, multiply and divide instructions

LAD / FBD	SCL	Description

	`out := in1 + in2;`
	`out := in1 - in2;`
	`out := in1 * in2;`
	`out := in1 / in2;`
		• ADD: Addition (IN1 + IN2 = OUT)
		• SUB: Subtraction (IN1 - IN2 = OUT)
		• MUL: Multiplication (IN1 * IN2 = OUT)
		• DIV: Division (IN1 / IN2 = OUT)
		An Integer division operation truncates the fractional part of the quotient to produce an integer output.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 39 Data types for the parameters (LAD and FBD)

Parameter	Data type[1]	Description
IN1, IN2	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Math operation inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Math operation output

[1] Parameters IN1, IN2, and OUTmust be the same data type.

To add an ADD or MUL input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

To remove an input, right-click on an input stub for one of the existing IN parameters (when there are more than the original two inputs) and select the "Delete" command.

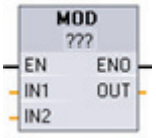
When enabled (EN = 1), the math instruction performs the specified operation on the input values (IN1 and IN2) and stores the result in the memory address specified by the output parameter (OUT). After the successful completion of the operation, the instruction sets ENO = 1.

Table 7- 40 ENO status

ENO	Description
1	No error
0	The Math operation result value would be outside the valid number range of the data type selected. The least significant part of the result that fits in the destination size is returned.
0	Division by 0 (IN2 = 0): The result is undefined and zero is returned.
0	Real/LReal: If one of the input values is NaN (not a number) then NaN is returned.
0	ADD Real/LReal: If both IN values are INF with different signs, this is an illegal operation and NaN is returned.
0	SUB Real/LReal: If both IN values are INF with the same sign, this is an illegal operation and NaN is returned.
0	MUL Real/LReal: If one IN value is zero and the other is INF, this is an illegal operation and NaN is returned.
0	DIV Real/LReal: If both IN values are zero or INF, this is an illegal operation and NaN is returned.

7.5.3 Modulo instruction

Table 7- 41 MOD instruction

LAD / FBD	SCL	Description
	`out := in1 MOD in2;`	You can use the MOD instruction to return the remainder of an integer division operation. The value at the IN1 input is divided by the value at the IN2 input and the remainder is returned at the OUT output.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 42 Data types for parameters

Parameter	Data type[1]	Description
IN1 and IN2	SInt, Int, DInt, USInt, UInt, UDInt, Constant	Modulo inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt	Modulo output

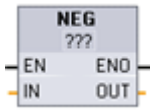
[1] The IN1, IN2, and OUTparameters must be the same data type.

Table 7- 43 ENO values

ENO	Description
1	No error
0	Value IN2 = 0, OUT is assigned the value zero

7.5.4 Negation instruction

Table 7- 44 NEG instruction

LAD / FBD	SCL	Description
	`-(in);`	The NEG instruction inverts the arithmetic sign of the value at parameter IN and stores the result in parameter OUT.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 45 Data types for parameters

Parameter	Data type[1]	Description
IN	SInt, Int, DInt, Real, LReal, Constant	Math operation input
OUT	SInt, Int, DInt, Real, LReal	Math operation output

[1] The IN and OUT parameters must be the same data type.

Table 7- 46 ENO status

ENO	Description
1	No error
0	The resulting value is outside the valid number range of the selected data type. Example for SInt: NEG (-128) results in +128 which exceeds the data type maximum.

7.5.5 Increment and decrement instructions

Table 7- 47 INC and DEC instructions

LAD / FBD	SCL	Description
	`in_out := in_out + 1;`	Increments a signed or unsigned integer number value: IN_OUT value +1 = IN_OUT value
	`in_out := in_out - 1;`	Decrements a signed or unsigned integer number value: IN_OUT value - 1 = IN_OUT value

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 48 Data types for parameters

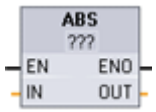
Parameter	Data type	Description
IN/OUT	SInt, Int, DInt, USInt, UInt, UDInt	Math operation input and output

Table 7- 49 ENO status

ENO	Description
1	No error
0	The resulting value is outside the valid number range of the selected data type. Example for SInt: INC (+127) results in +128, which exceeds the data type maximum.

7.5.6 Absolute value instruction

Table 7- 50 ABS instruction

LAD / FBD	SCL	Description
	`out := ABS(in);`	Calculates the absolute value of a signed integer or real number at parameter IN and stores the result in parameter OUT.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 51 Data types for parameters

Parameter	Data type[1]	Description
IN	SInt, Int, DInt, Real, LReal	Math operation input
OUT	SInt, Int, DInt, Real, LReal	Math operation output

[1] The IN and OUT parameters must be the same data type.

Table 7- 52 ENO status

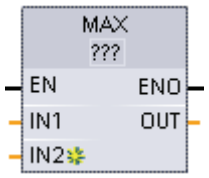
ENO	Description
1	No error
0	The math operation result value is outside the valid number range of the selected data type. Example for SInt: ABS (-128) results in +128 which exceeds the data type maximum.

7.5.7 Minimum and Maximum instructions

Table 7- 53 MIN and MAX instructions

LAD / FBD	SCL	Description

	`out:= MIN(`
	`in1:=_variant_in_,`
	`in2:=_variant_in_`
	`[,...in32]);`
		The MIN instruction compares the value of two parameters IN1 and IN2 and assigns the minimum (lesser) value to parameter OUT.

	`out:= MAX(`
	`in1:=_variant_in_,`
	`in2:=_variant_in_`
	`[,...in32]);`
		The MAX instruction compares the value of two parameters IN1 and IN2 and assigns the maximum (greater) value to parameter OUT.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 54 Data types for the parameters

Parameter	Data type[1]	Description
IN1, IN2 [...IN32]	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Math operation inputs (up to 32 inputs)
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Math operation output

[1] The IN1, IN2, and OUT parameters must be the same data type.

To add an input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

To remove an input, right-click on an input stub for one of the existing IN parameters (when there are more than the original two inputs) and select the "Delete" command.

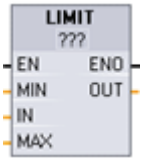
Table 7- 55 ENO status

ENO	Description
1	No error
0	For Real data type only: • One or more inputs is not a real number (NaN). • The resulting OUT is +/- INF (infinity).

7.5.8 Limit instruction

Table 7- 56 LIMIT instruction

LAD / FBD	SCL	Description

	`LIMIT(MIN:=_variant_in_,`
	`IN:=_variant_in_,`
	`MAX:=_variant_in_,`
	`OUT:=_variant_out_);`
		The Limit instruction tests if the value of parameter IN is inside the value range specified by parameters MIN and MAX and if not, clamps the value at MIN or MAX.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 57 Data types for the parameters

Parameter	Data type[1]	Description
MIN, IN, and MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Math operation inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Math operation output

1 The MIN, IN, MAX, and OUT parameters must be the same data type.

If the value of parameter IN is within the specified range, then the value of IN is stored in parameter OUT. If the value of parameter IN is outside of the specified range, then the OUT value is the value of parameter MIN (if the IN value is less than the MIN value) or the value of parameter MAX (if the IN value is greater than the MAX value).

Table 7- 58 ENO status

ENO	Description
1	No error
0	Real: If one or more of the values for MIN, IN and MAX is NaN (Not a Number), then NaN is returned.
0	If MIN is greater than MAX, the value IN is assigned to OUT.

SCL examples:

- MyVal := LIMIT(MIN:=10,IN:=53, MAX:=40); //Result: MyVal = 40
- MyVal := LIMIT(MIN:=10,IN:=37, MAX:=40); //Result: MyVal = 37
- MyVal := LIMIT(MIN:=10,IN:=8, MAX:=40); //Result: MyVal = 10

7.5.9 Floating-point math instructions

You use the floating point instructions to program mathematical operations using a Real or LReal data type:

- SQR: Square (IN ^{2} = OUT)
- SQRT: Square root (√IN = OUT)
- LN: Natural logarithm (LN(IN) = OUT)

- EXP: Natural exponential (e ^{IN} =OUT), where base e = 2.71828182845904523536
- EXPT: General exponential (IN1 ^{IN2} = OUT)

  EXPT parameters IN1 and OUT are always the same data type, for which you must select Real or LReal. You can select the data type for the exponent parameter IN2 from among many data types.
- FRAC: Fraction (fractional part of floating point number IN = OUT)
- SIN: Sine (sin(IN radians) = OUT)
  ASIN: Inverse sine (arcsine(IN) = OUT radians), where the sin(OUT radians) = IN
- COS: Cosine (cos(IN radians) = OUT)
  ACOS: Inverse cosine (arccos(IN) = OUT radians), where the cos(OUT radians) = IN
- TAN: Tangent (tan(IN radians) = OUT)
  ATAN: Inverse tangent (arctan(IN) = OUT radians), where the tan(OUT radians) = IN

Table 7- 59 Examples of floating-point math instructions

LAD / FBD	SCL	Description

	`out := SQR(in);`
	or
	`out := in * in;`
		Square: IN ^{2} = OUT
		For example: If IN = 9, then OUT = 81.

	`out := EXPT(in1, in2);`
	or
	`out := in1 ** in2;`
		General exponential: IN1 ^{IN2} = OUT
		For example: If IN1 = 3 and IN2 = 2, then OUT = 9.

[1] For LAD and FBD: Click the "???" (by the instruction name) and select a data type from the drop-down menu.

[2] For SCL: You can also use the basic SCL math operators to create the mathematical expressions.

Table 7- 60 Data types for parameters

Parameter	Data type	Description
IN, IN1	Real, LReal, Constant	Inputs
IN2	SInt, Int, DInt, USInt, UInt,UDInt, Real, LReal, Constant	EXPT exponent input
OUT	Real, LReal	Outputs

Table 7- 61 ENO status

ENO	Instruction	Condition	Result (OUT)
1	All	No error	Valid result
0	SQR	Result exceeds valid Real/LReal range	+INF
		IN is +/- NaN (not a number)	+NaN
	SQRT	IN is negative	-NaN

ENO	Instruction	Condition	Result (OUT)
		IN is +/- INF (infinity) or +/- NaN	+/- INF or +/- NaN
	LN	IN is 0.0, negative, -INF, or -NaN	-NaN
		IN is +INF or +NaN	+INF or +NaN
	EXP	Result exceeds valid Real/LReal range	+INF
		IN is +/- NaN	+/- NaN
	SIN, COS, TAN	IN is +/- INF or +/- NaN	+/- INF or +/- NaN
	ASIN, ACOS	IN is outside valid range of -1.0 to +1.0	+NaN
		IN is +/- NaN	+/- NaN
	ATAN	IN is +/- NaN	+/- NaN
	FRAC	IN is +/- INF or +/- NaN	+NaN
	EXPT	IN1 is +INF and IN2 is not -INF	+INF
		IN1 is negative or -INF	+NaN if IN2 is Real/LReal, -INF otherwise
		IN1 or IN2 is +/- NaN	+NaN
		IN1 is 0.0 and IN2 is Real/LReal (only)	+NaN

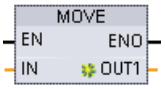
7.6 Move

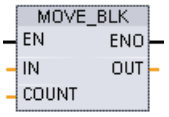
7.6.1 Move and block move instructions

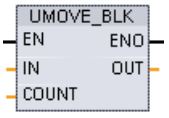
Use the Move instructions to copy data elements to a new memory address and convert from one data type to another. The source data is not changed by the move process.

- The MOVE instruction copies a single data element from the source address specified by the IN parameter to the destination addresses specified by the OUT parameter.
- The MOVE_BLK and UMOVE_BLK instructions have an additional COUNT parameter. The COUNT specifies how many data elements are copied. The number of bytes per element copied depends on the data type assigned to the IN and OUT parameter tag names in the PLC tag table.

Table 7- 62 MOVE, MOVE_BLK and UMOVE_BLK instructions

LAD / FBD	SCL	Description
	`out1 := in;`	Copies a data element stored at a specified address to a new address or multiple addresses.[1]

	`out := MOVE_BLK(`
	`  in:=_variant_in,`
	`  count:=_uint_in,`
	`  out=>_variant_out);`
		Interruptible move that copies a block of data elements to a new address.

	`out := UMOVE_BLK(`
	`  in:=_variant_in,`
	`  count:=_uint_in,`
	`  out=>_variant_out);`
		Uninterruptible move that copies a block of data elements to a new address.

[1] MOVE instruction: To add another output in LAD or FBD, click the "Create" icon by the output parameter. For SCL, use multiple assignment statements. You might also use one of the loop constructions.

Table 7- 63 Data types for the MOVE instruction

Parameter	Data type	Description
IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Char, Array, Struct, DTL, Time	Source address
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Char, Array, Struct, DTL, Time	Destination address

To add MOVE outputs, click the "Create" icon or right-click on an output stub for one of the existing OUT parameters and select the "Insert output" command.

To remove an output, right-click on an output stub for one of the existing OUT parameters (when there are more than the original two outputs) and select the "Delete" command.

Table 7- 64 Data types for the MOVE_BLK and UMOVE_BLK instructions

Parameter	Data type	Description
IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal Byte, Word, DWord	Source start address
COUNT	UInt	Number of data elements to copy
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord	Destination start address

Note

Rules for data copy operations

- To copy the Bool data type, use SET_BF, RESET_BF, R, S, or output coil (LAD) (Page 166)
- To copy a single elementary data type, use MOVE
- To copy an array of an elementary data type, use MOVE_BLK or UMOVE_BLK
- To copy a structure, use MOVE
- To copy a string, use S_MOVE (Page 240)
- To copy a single character in a string, use MOVE
- The MOVE_BLK and UMOVE_BLK instructions cannot be used to copy arrays or structures to the I, Q, or M memory areas.

MOVE_BLK and UMOVE_BLK instructions differ in how interrupts are handled:

- Interrupt events are **queued and processed** during MOVE_BLK execution. Use the MOVE_BLK instruction when the data at the move destination address is not used within an interrupt OB subprogram or, if used, the destination data does not have to be consistent. If a MOVE_BLK operation is interrupted, then the last data element moved is complete and consistent at the destination address. The MOVE_BLK operation is resumed after the interrupt OB execution is complete.
- Interrupt events are **queued but not processed** until UMOVE_BLK execution is complete. Use the UMOVE_BLK instruction when the move operation must be completed and the destination data consistent, before the execution of an interrupt OB subprogram. For more information, see the section on data consistency (Page 143).

ENO is always true following execution of the MOVE instruction.

Table 7- 65 ENO status

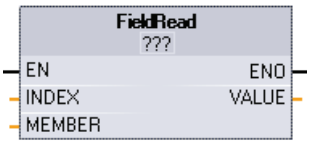
ENO	Condition	Result
1	No error	All COUNT elements were successfully copied.
0	Either the source (IN) range or the destination (OUT) range exceeds the available memory area.	Elements that fit are copied. No partial elements are copied.

7.6.2 FieldRead and FieldWrite instructions

Note

STEP 7 V10.5 **did not support** a variable reference as an array index or multi-dimensional arrays. The FieldRead and FieldWrite instructions were used to provide variable array index operations for a one-dimensional array. STEP 7 V11 **does support** a variable as an array index and multi-dimensional arrays. FieldRead and FieldWrite are included in STEP 7 V11 for backward compatibility with programs that have used these instructions.

Table 7- 66 FieldRead and FieldWrite instructions

LAD / FBD	SCL	Description
	`value := member[index];`	FieldRead reads the array element with the index value INDEX from the array whose first element in specified by the MEMBER parameter. The value of the array element is transferred to the location specified at the VALUE parameter.
	`member[index] := value;`	WriteField transfers the value at the location specified by the VALUE parameter to the array whose first element is specified by the MEMBER parameter. The value is transferred to the array element whose array index is specified by the INDEX parameter.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 67 Data types for parameters

Parameter and type		Data type	Description
Index	Input	DInt	The index number of the array element to be read or written to
Member [1]	Input	Array element types: Bool, Byte, Word, DWord, Char, SInt, Int, Dint, USInt, UInt, UDInt, Real, LReal	Location of the first element in a one-dimension array defined in a global data block or block interface. For example: If the array index is specified as [-2..4], then the index of the first element is -2 and not 0.
Value [1]	Out	Bool, Byte, Word, DWord, Char, SInt, Int, Dint, USInt, UInt, UDInt, Real, LReal	Location to which the specified array element is copied (FieldRead) Location of the value that is copied to the specified array element (FieldWrite)

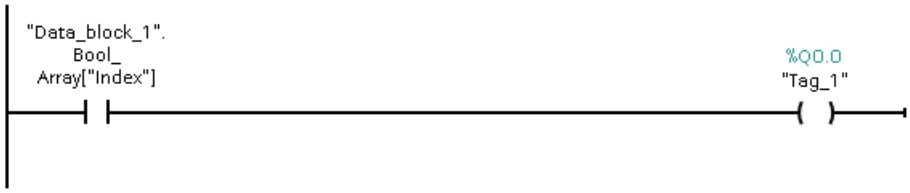
[1] The data type of the array element specified by the MEMBER parameter and the VALUE parameter must have the same data type.

The enable output ENO = 0, if one of the following conditions applies:

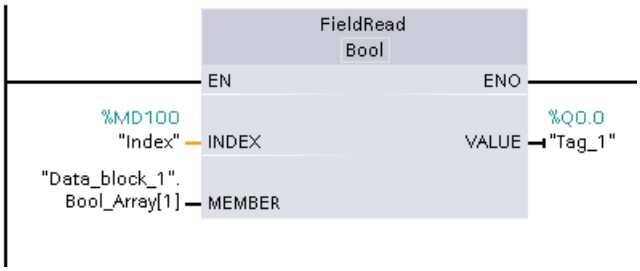
- The EN input has signal state "0"
- The array element specified at the INDEX parameter is not defined in the array referenced at MEMBER parameter
- Errors such as an overflow occur during processing

Accessing data by array indexing

To access elements of an array with a variable, simply use the variable as an array index in your program logic. For example, the network below sets an output based on the Boolean value of an array of Booleans in "Data_block_1" referenced by the PLC tag "Index".



The logic with the variable array index is equivalent to the former method using the FieldRead instruction:



FieldWrite and FieldRead instructions can be replaced with variable array indexing logic.

SCL has no FieldRead or FieldWrite instructions, but supports indirect addressing of an array with a variable:

```
#Tag_1 := "Data_block_1".Bool_Array[#Index];
```

7.6.3 Fill instructions

Table 7- 68 FILL_BLK and UFILL_BLK instructions

LAD / FBD	SCL	Description

	`out := FILL_BLK(`
	`in:=_variant_in,`
	`count:=int,`
	`out=>_variant_out);`
		Interruptible fill instruction: Fills an address range with copies of a specified data element

	`out := UFILL_BLK(`
	`in:=_variant_in,`
	`count:=int`
	`out=>_variant_out);`
		Uninterruptible fill instruction: Fills an address range with copies of a specified data element

Table 7- 69 Data types for parameters

Parameter	Data type	Description
IN	SInt, Int, DIntT, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord	Data source address
COUNT	USInt, UInt	Number of data elements to copy
OUT	SInt, Int, DIntT, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord	Data destination address

Note

Rules for data fill operations

- To fill with the BOOL data type, use SET_BF, RESET_BF, R, S, or output coil (LAD)
- To fill with a single elementary data type, use MOVE
- To fill an array with an elementary data type, use FILL_BLK or UFILL_BLK
- To fill a single character in a string, use MOVE
- The FILL_BLK and UFILL_BLK instructions cannot be used to fill arrays in the I, Q, or M memory areas.

The FILL_BLK and UFILL_BLK instructions copy the source data element IN to the destination where the initial address is specified by the parameter OUT. The copy process repeats and a block of adjacent addresses is filled until the number of copies is equal to the COUNT parameter.

FILL_BLK and UFILL_BLK instructions differ in how interrupts are handled:

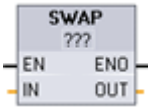
- Interrupt events are **queued and processed** during FILL_BLK execution. Use the FILL_BLK instruction when the data at the move destination address is not used within an interrupt OB subprogram or, if used, the destination data does not have to be consistent.
- Interrupt events are **queued but not processed** until UFILL_BLK execution is complete. Use the UFILL_BLK instruction when the move operation must be completed and the destination data consistent, before the execution of an interrupt OB subprogram.

Table 7- 70 ENO status

ENO	Condition	Result
1	No error	The IN element was successfully copied to all COUNT destinations.
0	The destination (OUT) range exceeds the available memory area	Elements that fit are copied. No partial elements are copied.

7.6.4 Swap instruction

Table 7- 71 SWAP instruction

LAD / FBD	SCL	Description
	`out := SWAP(in);`	Reverses the byte order for two-byte and four-byte data elements. No change is made to the bit order within each byte. ENO is always TRUE following execution of the SWAP instruction.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 72 Data types for the parameters

Parameter	Data type	Description
IN	Word, DWord	Ordered data bytes IN
OUT	Word, DWord	Reverse ordered data bytes OUT

Example 1	Parameter IN = MB0 (before execution)		Parameter OUT = MB4, (after execution)	
Address	MB0	MB1	MB4	MB5
W#16#1234	12	34	34	12
WORD	MSB	LSB	MSB	LSB

Example 2	Parameter IN = MB0 (before execution)				Parameter OUT = MB4, (after execution)			
Address	MB0	MB1	MB2	MB3	MB4	MB5	MB6	MB7
DW#16# 12345678	12	34	56	78	78	56	34	12
DWORD	MSB			LSB	MSB			LSB

7.7 Convert

7.7.1 CONV instruction

Table 7- 73 Convert (CONV) instruction

LAD / FBD	SCL	Description
	`out := <data type in>_TO_<data type out>(in);`	Converts a data element from one data type to another data type.

1 For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

2 For SCL: Construct the conversion instruction by identifying the data type for the input parameter (in) and output parameter (out). For example, DWORD_TO_REAL converts a DWord value to a Real value.

Table 7- 74 Data types for the parameters

Parameter	Data type	Description
IN	Bit string[1], SInt, USInt, Int, UInt, DInt, UDInt, Real, LReal, BCD16, BCD32	Input value
OUT	Bit string[1], SInt, USInt, Int, UInt, DInt, UDInt, Real, LReal, BCD16, BCE32	Input value converted to a new data type

1 The instruction does not allow you to select Bit strings (Byte, Word, DWord). To enter an operand of data type Byte, Word, or DWord for a parameter of the instruction, select an unsigned integer with the same bit length. For example, select USInt for a Byte, UInt for a Word, or UDInt for a DWord.

After you select the (convert from) data type, a list of possible conversions is shown in the (convert to) dropdown list. Conversions from and to BCD16 are restricted to the Int data type. Conversions from and to BCD32 are restricted to the DInt data type.

Table 7- 75 ENO status

ENO	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN
0	Result exceeds valid range for OUT data type	OUT is set to the least-significant bytes of IN

7.7.2 Conversion instructions for SCL

Conversion instructions for SCL

Table 7- 76 Conversion from a Byte, Word, or DWord

Data type	Instruction	Result
Byte	BYTE_TO_WORD, BYTE_TO_DWORD	The value is transferred to the low byte of the target data type.
	BYTE_TO_SINT, BYTE_TO_USINT	The value is transferred to the target data type.
	BYTE_TO_INT, BYTE_TO_UINT, BYTE_TO_DINT, BYTE_TO_UDINT	The value is transferred to the least significant byte of the target data type.
Word	WORD_TO_BYTE	The low byte of the source value is transferred to the target data type.
	WORD _TO_DWORD	The value is transferred to the low byte of the target data type.
	WORD _TO_SINT, WORD _TO_USINT	The low byte of the source value is transferred to the target data type.
	WORD _TO_INT, WORD _TO_UINT	The value is transferred to the target data type.
	WORD _TO_DINT, WORD _TO_UDINT	The value is transferred to the low byte of the target data type.
DWord	DWORD_TO_BYTE, DWORD_TO_WORD, DWORD_TO_SINT, DWORD_TO_USINT, DWORD_TO_INT, DWORD_TO_UINT	The low byte of the source value is transferred to the target data type.
	DWORD_TO_DINT, DWORD_TO_UDINT, DWORD_TO_REAL	The value is transferred to the target data type.

Table 7- 77 Conversion from a short integer (SInt or USInt)

Data type	Instruction	Result
SInt	SINT_TO_BYTE	The value is transferred to the target data type.
	SINT_TO_WORD, SINT_TO_DWORD, SINT_TO_INT, SINT_TO_DINT	The value is transferred to the low byte of the target data type.
	SINT_TO_USINT, SINT_TO_UINT, SINT_TO_UDINT, SINT_TO_REAL, SINT_TO_LREAL, SINT_TO_CHAR, SINT_TO_STRING	The value is converted.
USInt	USINT_TO_BYTE	The value is transferred to the target data type.
	USINT_TO_WORD, USINT_TO_DWORD, USINT_TO_INT, USINT_TO_UINT, USINT_TO_DINT, USINT_TO_UDINT	The value is transferred to the low byte of the target data type.
	USINT_TO_SINT, USINT_TO_REAL, USINT_TO_LREAL, USINT_TO_CHAR, USINT_TO_STRING	The value is converted.

Table 7- 78 Conversion from an integer (Int or UInt)

Data type	instruction	Result
Int	INT_TO_BYTE, INT_TO_DWORD, INT_TO_SINT, INT_TO_USINT, INT_TO_UINT, INT_TO_UDINT, INT_TO_REAL, INT_TO_LREAL, INT_TO_CHAR, INT_TO_STRING	The value is converted.
	INT_TO_WORD	The value is transferred to the target data type.
	INT_TO_DINT	The value is transferred to the low byte of the target data type.
UInt	UINT_TO_BYTE, UINT_TO_SINT, UINT_TO_USINT, UINT_TO_INT, UINT_TO_REAL, UINT_TO_LREAL, UINT_TO_CHAR, UINT_TO_STRING	The value is converted.
	UINT_TO_WORD, UINT_TO_DATE	The value is transferred to the target data type.
	UINT_TO_DWORD, UINT_TO_DINT, UINT_TO_UDINT	The value is transferred to the low byte of the target data type.

Table 7- 79 Conversion from a double integer (Dint or UDInt)

Data type	Instruction	Result
DInt	DINT_TO_BYTE, DINT_TO_WORD, DINT_TO_SINT, DINT_TO_USINT, DINT_TO_INT, DINT_TO_UINT, DINT_TO_UDINT, DINT_TO_REAL, DINT_TO_LREAL, DINT_TO_CHAR, DINT_TO_STRING	The value is converted.
	DINT_TO_DWORD, DINT_TO_TIME	The value is transferred to the target data type.
UDInt	UDINT_TO_BYTE, UDINT_TO_WORD, UDINT_TO_SINT, UDINT_TO_USINT, UDINT_TO_INT, UDINT_TO_UINT, UDINT_TO_DINT, UDINT_TO_REAL, UDINT_TO_LREAL, UDINT_TO_CHAR, UDINT_TO_STRING	The value is converted.
	UDINT_TO_DWORD, UDINT_TO_TOD	The value is transferred to the target data type.

Table 7- 80 Conversion from a Real number (Real or LReal)

Data type	Instruction	Result
Real	REAL_TO_DWORD, REAL_TO_LREAL	The value is transferred to the target data type.
	REAL_TO_SINT, REAL_TO_USINT, REAL_TO_INT, REAL_TO_UINT, REAL_TO_DINT, REAL_TO_UDINT, REAL_TO_TOD, REAL_TO_STRING	The value is converted.
LReal	LREAL_TO_SINT, LREAL_TO_USINT, LREAL_TO_INT, LREAL_TO_UINT, LREAL_TO_DINT, LREAL_TO_UDINT, LREAL_TO_REAL, REAL_TO_STRING	The value is converted.

Table 7- 81 Conversion from Time, DTL, TOD or Date

Data type	Instruction	Result
Time	`TIME_TO_DINT`	The value is transferred to the target data type.
DTL	`DTL_TO_DATE, DTL_TO_TOD`	The value is converted.
TOD	`TOD_TO_UDINT`	The value is converted.
Date	`DATE_TO_UINT`	The value is converted.

Table 7- 82 Conversion from a Char or String

Data type	Instruction	Result
Char	`CHAR_TO_SINT, CHAR_TO_USINT, CHAR_TO_INT, CHAR_TO_UINT, CHAR_TO_DINT, CHAR_TO_UDINT`	The value is converted.
	`CHAR_TO_STRING`	The value is transferred to the first character of the string.
String	`STRING_TO_SINT, STRING_TO_USINT, STRING_TO_INT, STRING_TO_UINT, STRING_TO_DINT, STRING_TO_UDINT, STRING_TO_REAL, STRING_TO_LREAL`	The value is converted.
	`STRING_TO_CHAR`	The first character of the string is copied to the Char.

7.7.3 Round and truncate instructions

Table 7- 83 ROUND and TRUNC instructions

LAD / FBD	SCL	Description
	`out := ROUND (in);`	Converts a real number to an integer. The real number fraction is rounded to the nearest integer value (IEEE - round to nearest). If the number is exactly one-half the span between two integers (for example, 10.5), then the number is rounded to the even integer. For example: • ROUND (10.5) = 10 • ROUND (11.5) = 12
	`out := TRUNC(in);`	TRUNC converts a real number to an integer. The fractional part of the real number is truncated to zero (IEEE - round to zero).

[1] For LAD and FBD: Click the "???" (by the instruction name) and select a data type from the drop-down menu.

Table 7- 84 Data types for the parameters

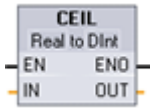
Parameter	Data type	Description
IN	Real, LReal	Floating point input
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Rounded or truncated output

Table 7- 85 ENO status

ENO	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN

7.7.4 Ceiling and floor instructions

Table 7- 86 CEIL and FLOOR instructions

LAD / FBD	SCL	Description
	`out := CEIL(in);`	Converts a real number (Real or LReal) to the closest integer greater than or equal to the selected real number (IEEE "round to +infinity").
	`out := FLOOR(in);`	Converts a real number (Real or LReal) to the closest integer smaller than or equal to the selected real number (IEEE "round to -infinity").

[1] For LAD and FBD: Click the "???" (by the instruction name) and select a data type from the drop-down menu.

Table 7- 87 Data types for the parameters

Parameter	Data type	Description
IN	Real, LReal	Floating point input
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Converted output

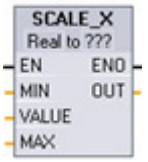
Table 7- 88 ENO status

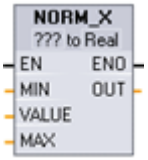
ENO	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN

7.7.5 Scale and normalize instructions

Table 7- 89 SCALE_X and NORM_X instructions

LAD / FBD	SCL	Description

	`out :=SCALE_X(min:=_in_,`
	`value:=_in_,`
	`max:=_in_);`
	or
	`out := value (max - min) + min;`
		Scales the normalized real parameter VALUE where (0.0 <= VALUE <= 1.0) in the data type and value range specified by the MIN and MAX parameters:
		OUT = VALUE (MAX - MIN) + MIN

	`out :=NORM_X(min:=_in_,`
	`value:=_in_,`
	`max:=_in_);`
	or
	`out := (value - min) / (max - min);`
		Normalizes the parameter VALUE inside the value range specified by the MIN and MAX parameters:
		OUT = (VALUE - MIN) / (MAX - MIN),
		where (0.0 <= OUT <= 1.0)

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 90 Data types for the parameters

Parameter	Data type[1]	Description
MIN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input minimum value for range
VALUE	SCALE_X: Real, LReal NORM_X: SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input value to scale or normalize
MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input maximum value for range
OUT	SCALE_X: SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal NORM_X: Real, LReal	Scaled or normalized output value

1 For SCALE_X: Parameters MIN, MAX, and OUTmust be the same data type.
For NORM_X: Parameters MIN, VALUE, and MAXmust be the same data type.

Note

SCALE_X parameter VALUE should be restricted to (0.0 <= VALUE <= 1.0)

If parameter VALUE is less than 0.0 or greater than 1.0:

- The linear scaling operation can produce OUT values that are less than the parameter MIN value or above the parameter MAX value for OUT values that fit within the value range of the OUT data type. SCALE_X execution sets ENO = TRUE for these cases.
- It is possible to generate scaled numbers that are not within the range of the OUT data type. For these cases, the parameter OUT value is set to an intermediate value equal to the least-significant portion of the scaled real number prior to final conversion to the OUT data type. SCALE_X execution sets ENO = FALSE in this case.

NORM_X parameter VALUE should be restricted to (MIN <= VALUE <= MAX)

If parameter VALUE is less than MIN or greater than MAX, the linear scaling operation can produce normalized OUT values that are less than 0.0 or greater than 1.0. NORM_X execution sets ENO = TRUE in this case.

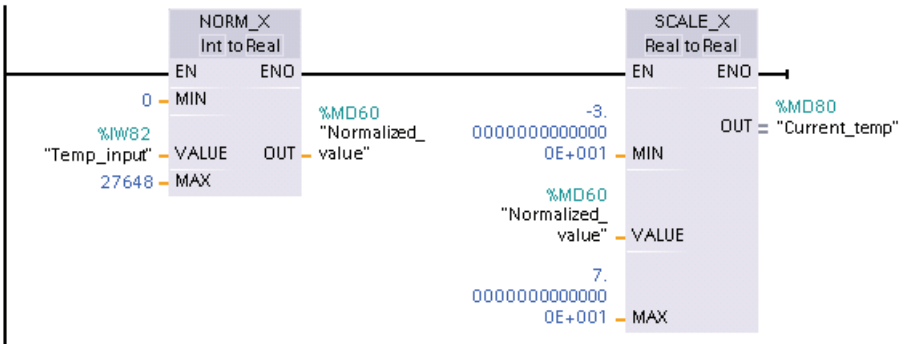
Table 7- 91 ENO status

ENO	Condition	Result OUT
1	No error	Valid result
0	Result exceeds valid range for the OUT data type	Intermediate result: The least-significant portion of a real number prior to final conversion to the OUT data type.
0	Parameters MAX <= MIN	SCALE_X: The least-significant portion of the Real number VALUE to fill up the OUT size. NORM_X: VALUE in VALUE data type extended to fill a double word size.
0	Parameter VALUE = +/- INF or +/- NaN	VALUE is written to OUT

Example (LAD): normalizing and scaling an analog input value

An analog input from an analog signal module or signal board using input in current is in the range 0 to 27648 for valid values. Suppose an analog input represents a temperature where the 0 value of the analog input represents -30.0 degrees C and 27648 represents 70.0 degrees C.

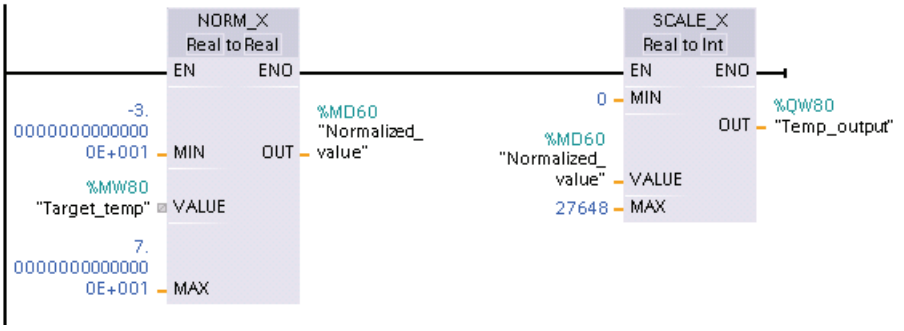
To transform the analog value to the corresponding engineering units, normalize the input to a value between 0.0 and 1.0, and then scale it between -30.0 and 70.0. The resulting value is the temperature represented by the analog input in degrees C:



Note that if the analog input was from an analog signal module or signal board using voltage, the MIN value for the NORM_X instruction would be -27648 instead of 0.

Example (LAD): normalizing and scaling an analog output value

An analog output to be set in an analog signal module or signal board using output in current must be in the range 0 to 27648 for valid values. Suppose an analog output represents a temperature setting where the 0 value of the analog input represents -30.0 degrees C and 27648 represents 70.0 degrees C. To convert a temperature value in memory that is between -30.0 and 70.0 to a value for the analog output in the range 0 to 27648, you must normalize the value in engineering units to a value between 0.0 and 1.0, and then scale it to the range of the analog output, 0 to 27648:



Note that if the analog output was for an analog signal module or signal board using voltage, the MIN value for the SCALE_X instruction would be -27648 instead of 0.

Additional information on analog input representations (Page 708) and analog output representations (Page 708) in both voltage and current can be found in the Technical Specifications.

7.8 Program control

7.8.1 Overview of SCL program control statements

Structured Control Language (SCL) provides three types of program control statements for structuring your user program:

- Selective statements: A selective statement enables you to direct program execution into alternative sequences of statements.
- Loops: You can control loop execution using iteration statements. An iteration statement specifies which parts of a program should be iterated depending on certain conditions.
- Program jumps: A program jump means an immediate jump to a specified jump destination and therefore to a different statement within the same block.

These program control statements use the syntax of the PASCAL programming language.

Table 7- 92 Types of SCL program control statements

Program control statement		Description
Selective	IF-THEN statement (Page 210)	Enables you to direct program execution into one of two alternative branches, depending on a condition being TRUE or FALSE
	CASE statement (Page 211)	Enables the selective execution into 1 of *n* alternative branches, based on the value of a variable
Loop	FOR statement (Page 212)	Repeats a sequence of statements for as long as the control variable remains within the specified value range
	WHILE-DO statement (Page 213)	Repeats a sequence of statements while an execution condition continues to be satisfied
	REPEAT-UNTIL statement (Page 214)	Repeats a sequence of statements until a terminate condition is met
Program jump	CONTINUE statement (Page 214)	Stops the execution of the current loop iteration
	EXIT statement (Page 215)	Exits a loop at any point regardless of whether the terminate condition is satisfied or not
	GOTO statement (Page 216)	Causes the program to jump immediately to a specified label
	IF-THEN statement (Page 210)	Causes the program to exit the block currently being executed and to return to the calling block

See also

RETURN statement (Page 216)

7.8.2 IF-THEN statement

The IF-THEN statement is a conditional statement that controls program flow by executing a group of statements, based on the evaluation of a Bool value of a logical expression. You can also use brackets to nest or structure the execution of multiple IF-THEN statements.

Table 7- 93 Elements of the IF-THEN statement

SCL	Description

`IF "condition" THEN`
`  statement_A;`
`  statement_B;`
`  statement_C;`
`  ;`
	If "condition" is TRUE or 1, then execute the following statements until encountering the END_IF statement.
	If "condition" is FALSE or 0, then skip to END_IF statement (unless the program includes optional ELSIF or ELSE statements).

`[ELSIF "condition-n" THEN`
`  statement_N;`
`  ;]`
	The optional ELSEIF[1] statement provides additional conditions to be evaluated. For example: If "condition" in the IF-THEN statement is FALSE, then the program evaluates "condition-n". If "condition-n" is TRUE, then execute "statement_N".

`[ELSE`
`  statement_X;`
`  ;]`
	The optional ELSE statement provides statements to be executed when the "condition" of the IF-THEN statement is FALSE.

`END_IF;`	The END_IF statement terminates the IF-THEN instruction.

[1] You can include multiple ELSIF statements within one IF-THEN statement.

Table 7- 94 Variables for the IF-THEN statement

Variables	Description
"condition"	Required. The logical expression is either TRUE (1) or FALSE (0).
"statement_A"	Optional. One or more statements to be executed when "condition" is TRUE.
"condition-n"	Optional. The logical expression to be evaluated by the optional ELSIF statement.
"statement_N"	Optional. One or more statements to be executed when "condition-n" of the ELSIF statement is TRUE.
"statement_X"	Optional. One or more statements to be executed when "condition" of the IF-THEN statement is FALSE.

An IF statement is executed according to the following rules:

- The first sequence of statements whose logical expression = TRUE is executed. The remaining sequences of statements are not executed.
- If no Boolean expression = TRUE, the sequence of statements introduced by ELSE is executed (or no sequence of statements if the ELSE branch does not exist).
- Any number of ELSIF statements can exist.

Note

Using one or more ELSIF branches has the advantage that the logical expressions following a valid expression are no longer evaluated in contrast to a sequence of IF statements. The runtime of a program can therefore be reduced.

7.8.3 CASE statement

Table 7- 95 Elements of the CASE statement

SCL	Description

`CASE "Test_Value" OF`
`  "ValueList": Statement[; Statement, ...]`
`  "ValueList": Statement[; Statement, ...]`
`[ELSE`
`Else-statement[; Else-statement, ...]]`
`END_CASE;`
	The CASE statement executes one of several groups of statements, depending on the value of an expression.

Table 7- 96 Parameters

Parameter	Description
"Test_Value"	Required. Any numeric expression of data type Int
"ValueList"	Required. A single value or a comma-separated list of values or ranges of values. (Use two periods to define a range of values: 2..8) The following example illustrates the different variants of the value list: 1: Statement_A; 2, 4: Statement _B; 3, 5..7,9: Statement _C;
Statement	Required. One or more statements that are executed when "Test_Value" matches any value in the value list
Else-statement	Optional. One or more statements that are executed if no match with a value of the "ValueList" stated matches

The CASE statement is executed according to the following rules:

- The selection expression must return a value of the type Int.
- When a CASE statement is processed, the program checks whether the value of the selection expression is contained within a specified list of values. If a match is found, the statement component assigned to the list is executed.
- If no match is found, the program section following ELSE is executed or no statement is executed if the ELSE branch does not exist.

CASE statements can be nested. Each nested case statement must have an associated END_CASE statement.

```
CASE var1 OF
     1 : var2 := "A";
     2 : var2 := "B";
     CASE var1 OF
          65..90: var2 := "UpperCase";
          97..122: var2 := "LowerCase";
     END_CASE
ELSE
     var1:= "SpecialCharacter";
```

```
END_CASE
```

7.8.4 FOR statement

Table 7- 97 Elements of the FOR statement

SCL	Description

`FOR "control_variable" := "begin" TO "end"`
`[BY "increment"] DO`
`  statement;`
`  ;`
`END_FOR;`
	A FOR statement is used to repeat a sequence of statements as long as a control variable is within the specified range of values. The definition of a loop with FOR includes the specification of an initial and an end value. Both values must be the same type as the control variable.

Table 7- 98 Parameters

Parameter	Description
"control_variable"	Required. An integer (Int or DInt) that serves as a loop counter
"begin"	Required. Simple expression that specifies the initial value of the control variables
"end"	Required. Simple expression that determines the final value of the control variables
"increment"	Optional. Amount by which a "control variable" is changed after each loop. The "increment" has the same data type as "control variable". If the "increment" value is not specified, then the value of the run tags will be increased by 1 after each loop. You cannot change "increment" during the execution of the FOR statement.

The FOR statement executes as follows:

- At the start of the loop, the control variable is set to the initial value (initial assignment) and each time the loop iterates, it is incremented by the specified increment (positive increment) or decremented (negative increment) until the final value is reached.
- Following each run through of the loop, the condition is checked (final value reached) to establish whether or not it is satisfied. If the condition is satisfied, the sequence of statements is executed, otherwise the loop and with it the sequence of statements is skipped.

Rules for formulating FOR statements:

- The control variable may only be of the data type Int or DInt.
- You can omit the statement BY [increment]. If no increment is specified, it is automatically assumed to be +1.

To end the loop regardless of the state of the "condition" expression, use the EXIT statement (Page 215). The EXIT statement executes the statement immediately following the END_FOR statement.

Use the CONTINUE statement (Page 214) to skip the subsequent statements of a FOR loop and to continue the loop with the examination of whether the condition is met for termination.

7.8.5 WHILE-DO statement

Table 7- 99 WHILE statement

SCL	Description

`WHILE "condition" DO`
`  Statement;`
`  Statement;`
`  ...;`
`END_WHILE;`
	The WHILE statement performs a series of statements until a given condition is TRUE.
	You can nest WHILE loops. The END_WHILE statement refers to the last executed WHILE instruction.

Table 7- 100 Parameters

Parameter	Description
"condition"	Required. A logical expression that evaluates to TRUE or FALSE. (A "null" condition is interpreted as FALSE.)
Statement	Optional. One or more statements that are executed until the condition evaluates to TRUE.

Note

The WHILE statement evaluates the state of "condition" before executing any of the statements. To execute the statements at least one time regardless of the state of "condition", use the REPEAT statement.

The WHILE statement executes according to the following rules:

- Prior to each iteration of the loop body, the execution condition is evaluated.
- The loop body following DO iterates as long as the execution condition has the value TRUE.
- Once the value FALSE occurs, the loop is skipped and the statement following the loop is executed.

To end the loop regardless of the state of the "condition" expression, use the EXIT statement (Page 215). The EXIT statement executes the statement immediately following the END_WHILE statement

Use the CONTINUE statement to skip the subsequent statements of a WHILE loop and to continue the loop with the examination of whether the condition is met for termination.

7.8.6 REPEAT-UNTIL statement

Table 7- 101 REPEAT instruction

SCL	Description

`REPEAT`
`  Statement;`
`  ;`
`UNTIL "condition"`
`END_REPEAT`
	The REPEAT statement executes a group of statements until a given condition is TRUE.
	You can nest REPEAT loops. The END_REPEAT statement always refers to the last executed Repeat instruction.

Table 7- 102 Parameters

Parameter	Description
Statement	Optional. One or more statements that are executed until the condition is TRUE.
"condition"	Required. One or more expressions of the two following ways: A numeric expression or string expression that evaluates to TRUE or FALSE. A "null" condition is interpreted as FALSE.

Note

Before evaluating the state of "condition", the REPEAT statement executes the statements during the first iteration of the loop (even if "condition" is FALSE). To review the state of "condition" before executing the statements, use the WHILE statement.

To end the loop regardless of the state of the "condition" expression, use the EXIT statement (Page 215). The EXIT statement executes the statement immediately following the END_REPEAT statement

Use the CONTINUE statement (Page 214) to skip the subsequent statements of a REPEAT loop and to continue the loop with the examination of whether the condition is met for termination.

7.8.7 CONTINUE statement

Table 7- 103 CONTINUE statement

SCL	Description

`CONTINUE`
`  Statement;`
`  ;`
	The CONTINUE statement skips the subsequent statements of a program loop (FOR, WHILE, REPEAT) and continues the loop with the examination of whether the condition is met for termination. If this is not the case, the loop continues.

The CONTINUE statement executes according to the following rules:

- This statement immediately terminates execution of a loop body.
- Depending on whether the condition for repeating the loop is satisfied or not the body is executed again or the iteration statement is exited and the statement immediately following is executed.
- In a FOR statement, the control variable is incremented by the specified increment immediately after a CONTINUE statement.

Use the CONTINUE statement only within a loop. In nested loops CONTINUE always refers to the loop that includes it immediately. CONTINUE is typically used in conjunction with an IF statement.

If the loop is to exit regardless of the termination test, use the EXIT statement.

The following example shows the use of the CONTINUE statement to avoid a division-by-0 error when calculating the percentage of a value:

```
FOR x = 0 TO 10 DO
IF value[i] = 0 THEN CONTINUE; END_IF;
     p := part / value[i] * 100;
     s := INT_TO_STRING(p);
     percent=CONCAT(IN1:=s, IN2:="%");
END_FOR;
```

7.8.8 EXIT statement

Table 7- 104 Exit instruction

SCL	Description
EXIT	An EXIT statement is used to exit a loop (FOR, WHILE or REPEAT) at any point, regardless of whether the terminate condition is satisfied.

The EXIT statement executes according to the following rules:

- This statement causes the repetition statement immediately surrounding the exit statement to be exited immediately.
- Execution of the program is continued after the end of the loop (for example after END_FOR).

Use the EXIT statement within a loop. In nested loops, the EXIT statement returns the processing to the next higher nesting level.

```
FOR i = 0 TO 10 DO
CASE value[i, 0] OF
 1..10: value [i, 1]:="A";
 11..40:= value [i, 1]:="B";
 41..100:= value [i, 1]:="C";
ELSE
EXIT;
END_CASE;
END_FOR;
```

7.8.9 GOTO statement

Table 7- 105 GOTO statement

SCL	Description

`GOTO JumpLabel`
`Statement;`
`... ;`
`JumpLabel: Statement;`
	The GOTO statement skips over statements by jumping to a label in the same block.
	The jump label ("JumpLabel") and the GOTO statement must be in the same block. The name of a jump label can only be assigned once within a block. Each jump label can be the target of several GOTO statements.

It is not possible to jump to a loop section (FOR, WHILE or REPEAT). It is possible to jump from within a loop.

In the following example: Depending on the value of the "Tag_value" operand, the execution of the program resumes at the point defined by the corresponding jump label. If "Tag_value" equals 2, the program execution resumes at the jump label "MyLabel2" and skips "MyLabel1".

```
CASE "Tag_value" OF
1 : GOTO MyLabel1;
2 : GOTO MyLabel2;
ELSE GOTO MYLabel3
END_CASE;
MyLabel1: "Tag_1" := 1;
MyLabel2: "Tag_2" := 1;
MyLabel3: "Tag_4" := 1;
```

7.8.10 RETURN statement

Table 7- 106 Return instruction

SCL	Description
`Return;`	The Return instruction exits the code block being executed without conditions. Program execution returns to the calling block or to the operating system (when exiting an OB).

Example of a Return instruction:

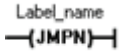
```
IF "Error" <> 0 THEN
Return;
END_IF
```

Note

After executing the last instruction, the code block automatically returns to the calling block. Do not insert a Return instruction at the end of the code block.

7.8.11 Jump and label instructions

Table 7- 107 JMP, JMPN, and LABEL instruction

LAD	FBD	SCL	Description
		See the GOTO (Page 216) statement.	If there is power flow to a JMP coil (LAD), or if the JMP box input is true (FBD), then program execution continues with the first instruction following the specified label.
			If there is no power flow to a JMPN coil (LAD), or if the JMPN box input is false (FBD), then program execution continues with the first instruction following the specified label.
			Destination label for a JMP or JMPN jump instruction.

[1] You create your label names by typing in the LABEL instruction directly. Use the parameter helper icon to select the available label names for the JMP and JMPN label name field. You can also type a label name directly into the JMP or JMPN instruction.

Table 7- 108 Data types for the parameters

Parameter	Data type	Description
Label_name	Label identifier	Identifier for Jump instructions and the corresponding jump destination program label

- Each label must be unique within a code block.
- You can jump within a code block, but you cannot jump from one code block to another code block.
- You can jump forward or backward.
- You can jump to the same label from more than one place in the same code block.

7.8.12 JMP_LIST instruction

Table 7- 109 JMP_LIST instruction

LAD / FBD	SCL	Description

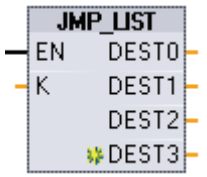
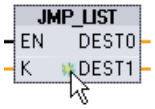
	`CASE k OF`
	`  0: GOTO dest0;`
	`  1: GOTO dest1;`
	`  2: GOTO dest2;`
	`  [n: GOTO destn;]`
	`END_CASE;`
		The JMP_LIST instruction acts as a program jump distributor to control the execution of program sections. Depending on the value of the K input, a jump occurs to the corresponding program label. Program execution continues with the program instructions that follow the destination jump label. If the value of the K input exceeds the number of labels - 1, then no jump occurs and processing continues with the next program network.

Table 7- 110 Data types for parameters

Parameter	Data type	Description
K	UInt	Jump distributor control value
DEST0, DEST1, .., DESTn.	Program Labels	Jump destination labels corresponding to specific K parameter values: If the value of K equals 0, then a jump occurs to the program label assigned to the DEST0 output. If the value of K equals 1, then a jump occurs to the program label assigned to the DEST1 output, and so on. If the value of the K input exceeds the (number of labels - 1), then no jump occurs and processing continues with the next program network.

For LAD and FBD: The JMP_LIST box is first placed in your program, there are two jump label outputs. You can add or delete jump destinations.

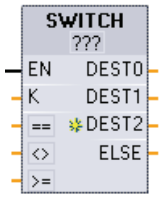


Click the create icon inside the box (on the left of the last DEST parameter) to add new outputs for jump labels.

- Right-click on an output stub and select the "Insert output" command.
- Right-click on an output stub and select the "Delete" command.

7.8.13 SWITCH instruction

Table 7- 111 SWITCH instruction

LAD / FBD	SCL	Description
	`Not available`	The SWITCH instruction acts as a program jump distributor to control the execution of program sections. Depending on the result of comparisons between the value of the K input and the values assigned to the specified comparison inputs, a jump occurs to the program label that corresponds to the first comparison test that is true. If none of the comparisons is true, then a jump to the label assigned to ELSE occurs. Program execution continues with the program instructions that follow the destination jump label.

1 For LAD and FBD: Click below the box name and select a data type from the drop-down menu.

2 For SCL: Use an IF-THEN set of comparisons.

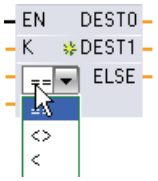
Table 7- 112 Data types for parameters

Parameter	Data type[1]	Description
K	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, TOD, Date	Common comparison value input
==, <>, <, <=, >. >=	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, TOD, Date	Separate comparison value inputs for specific comparison types
DEST0, DEST1, .., DESTn. ELSE	Program Labels	Jump destination labels corresponding to specific comparisons: The comparison input below and next to the K input is processed first and causes a jump to the label assigned to DEST0, if the comparison between the K value and this input is true. The next comparison test uses the next input below and causes a jump to the label assigned to DEST1, if the comparison is true, The remaining comparisons are processed similarly and if none of the comparisons are true, then a jump to the label assigned to the ELSE output occurs.

[1] The K input and comparison inputs (==, <>, <, <=, >, >=) must be the same data type.

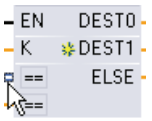
Adding inputs, deleting inputs, and specifying comparison types

When the LAD or FBD SWITCH box is first placed in your program there are two comparison inputs. You can assign comparison types and add inputs/jump destinations, as shown below.



Click a comparison operator inside the box and select a new operator from the drop-down list.

Click the create icon inside the box (to the left of the last DEST parameter) to add new comparison-destination parameters.



- Right-click on an input stub and select the "Insert input" command.
- Right-click on an input stub and select the "Delete" command.

Table 7- 113 SWITCH box data type selection and allowed comparison operations

Data type	Comparison	Operator syntax
Byte, Word, DWord	Equal	==
	Not equal	<>
SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Time, TOD, Date	Equal	==
	Not equal	<>
	Greater than or equal	>=
	Less than or equal	<=
	Greater than	>
	Less than	<

SWITCH box placement rules

- No LAD/FBD instruction connection in front of the compare input is allowed.
- There is no ENO output, so only one SWITCH instruction is allowed in a network and the SWITCH instruction must be the last operation in a network.

7.8.14 RET execution control instruction

The optional RET instruction is used to terminate the execution of the current block. If and only if there is power flow to the RET coil (LAD) or if the RET box input is true (FBD), then program execution of the current block will end at that point and instructions beyond the RET instruction will not be executed. If the current block is an OB, the "Return_Value" parameter is ignored. If the current block is a FC or FB, the value of the "Return_Value " parameter is passed back to the calling routine as the ENO value of the called box.

You are not required to use a RET instruction as the last instruction in a block; this is done automatically for you. You can have multiple RET instructions within a single block.

For SCL, see the RETURN (Page 216) statement.

Table 7- 114 Return_Value (RET) execution control instruction

LAD	FBD	SCL	Description
		`RETURN;`	Terminates the execution of the current block

Table 7- 115 Data types for the parameters

Parameter	Data type	Description
Return_Value	Bool	The "Return_value" parameter of the RET instruction is assigned to the ENO output of the block call box in the calling block.

Sample steps for using the RET instruction inside an FC code block:

1. Create a new project and add an FC:
2. Edit the FC:
   - Add instructions from the instruction tree.
   - Add a RET instruction, including one of the following for the "Return_Value" parameter:

     TRUE, FALSE, or a memory location that specifies the required return value.
   - Add more instructions.
3. Call the FC from MAIN [OB1].

The EN input on the FC box in the MAIN code block must be true to begin execution of the FC.

The value specified by the RET instruction in the FC will be present on the ENO output of the FC box in the MAIN code block following execution of the FC for which power flow to the RET instruction is true.

7.8.15 Re-trigger scan cycle watchdog instruction

Table 7- 116 RE_TRIGR instruction

LAD / FBD	SCL	Description
	`RE_TRIGR();`	RE_TRIGR (Re-trigger scan time watchdog) is used to extend the maximum time allowed before the scan cycle watchdog timer generates an error.

Use the RE_TRIGR instruction to restart the scan cycle timer during a single scan cycle. This has the effect of extending the allowed maximum scan cycle time by one maximum cycle time period, from the last execution of the RE_TRIGR function.

The CPU restricts the use of the RE_TRIGR instruction to the program cycle, for example, OB1 and functions that are called from the program cycle. This means that the watchdog timer is reset, and ENO = EN, if RE_TRIGR is called from any OB of the program cycle OB list.

ENO = FALSE and the watchdog timer is not reset if RE_TRIGR is executed from a start up OB, an interrupt OB, or an error OB.

Setting the PLC maximum cycle time

Configure the value for maximum scan cycle time in the Device configuration for "Cycle time".

Table 7- 117 Cycle time values

Cycle time monitor	Minimum value	Maximum value	Default value
Maximum cycle time	1 ms	6000 ms	150 ms

Watchdog timeout

If the maximum scan cycle timer expires before the scan cycle has been completed, an error is generated. If the error handling code block OB 80 is included in the user program, the CPU executes OB 80 where you may add program logic to create a special reaction. If OB 80 is not included, the first timeout condition is ignored.

If a second maximum scan time timeout occurs in the same program scan (2 times the maximum cycle time value), an error is triggered that causes the CPU to transition to STOP mode.

In STOP mode, your program execution stops while CPU system communications and system diagnostics continue.

7.8.16 Stop scan cycle instruction

Table 7- 118 STP instruction

LAD / FBD	SCL	Description
	`STP();`	STP (Stop scan cycle) puts the CPU in STOP mode. When the CPU is in STOP mode, the execution of your program and physical updates from the process image are stopped.

For more information see: Configuring the outputs on a RUN-to-STOP transition (Page 80).

If EN = TRUE, then the CPU goes to STOP mode, the program execution stops, and the ENO state is meaningless. Otherwise, EN = ENO = 0.

7.8.17 Get Error instructions

The get error instructions provide information about program block execution errors. If you add a GetError or GetErrorID instruction to your code block, you can handle program errors within your program block.

GetError

Table 7- 119 GetError instruction

LAD / FBD	SCL	Description
	Not available	Indicates that a local program block execution error has occurred and fills a predefined error data structure with detailed error information.

Table 7- 120 Data types for the parameters

Parameter	Data type	Description
ERROR	ErrorStruct	Error data structure: You can rename the structure, but not the members within the structure.

Table 7- 121 Elements of the ErrorStruct data structure

Structure components		Data type	Description
ERROR_ID		Word	Error ID
FLAGS		Byte	Shows if an error occurred during a block call. • 16#01: Error during a block call. • 16#00: No error during a block call.
REACTION		Byte	Default reaction: • 0: Ignore (write error), • 1: Continue with substitute value "0" (read error), • 2: Skip instruction (system error)
CODE_ADDRESS		CREF	Information about the address and type of block
	BLOCK_TYPE	Byte	Type of block where the error occurred: • 1: OB • 2: FC • 3: FB
	CB_NUMBER	UInt	Number of the code block
	OFFSET	UDInt	Reference to the internal memory
MODE		Byte	Access mode: Depending on the type of access, the following information can be output:

Mode	(A)	(B)	(C)	(D)	(E)
0					
1					Offset
2			Area		
3	Location	Scope		Number	
4			Area		Offset
5			Area	DB no.	Offset

Structure components		Data type	Description					
			6	PtrNo. /Acc		Area	DB no.	Offset
			7	PtrNo. / Acc	Slot No. / Scope	Area	DB no.	Offset
OPERAND_NUMBER		UInt	Operand number of the machine command					
POINTER_NUMBER_ LOCATION		UInt	(A) Internal pointer					
SLOT_NUMBER_SCOPE		UInt	(B) Storage area in internal memory					
DATA_ADDRESS		NREF	Information about the address of an operand					
	AREA	Byte	(C) Memory area: • L: 16#40 – 4E, 86, 87, 8E, 8F, C0 – CE • E: 16#81 • A: 16#82 • M: 16#83 • DB: 16#84, 85, 8A, 8B					
	DB_NUMBER	UInt	(D) Number of the data block					
	OFFSET	UDInt	(E) Relative address of the operand					

GetErrorID

Table 7- 122 GetErrorID instruction

LAD / FBD	SCL	Description
	Not available	Indicates that a program block execution error has occurred and reports the ID (identifier code) of the error.

Table 7- 123 Data types for the parameters

Parameter	Data type	Description
ID	Word	Error identifier values for the ErrorStruct ERROR_ID member

Table 7- 124 Error_ID values

ERROR_ID Hexadecimal	ERROR_ID Decimal	Program block execution error
0	0	No error
2503	9475	Uninitialized pointer error
2522	9506	Operand out of range read error
2523	9507	Operand out of range write error
2524	9508	Invalid area read error

ERROR_ID Hexadecimal	ERROR_ID Decimal	Program block execution error
2525	9509	Invalid area write error
2528	9512	Data alignment read error (incorrect bit alignment)
2529	9513	Data alignment write error (incorrect bit alignment)
2530	9520	DB write protected
253A	9530	Global DB does not exist
253C	9532	Wrong version or FC does not exist
253D	9533	Instruction does not exist
253E	9534	Wrong version or FB does not exist
253F	9535	Instruction does not exist
2575	9589	Program nesting depth error
2576	9590	Local data allocation error
2942	10562	Physical input point does not exist
2943	10563	Physical output point does not exist

Operation

By default, the CPU responds to a block execution error by logging an error in the diagnostics buffer. However, if you place one or more GetError or GetErrorID instructions within a code block, this block is now set to handle errors within the block. In this case, the CPU does not log an error in the diagnostics buffer. Instead, the error information is reported in the output of the GetError or GetErrorID instruction. You can read the detailed error information with the GetError instruction, or read just the error identifier with GetErrorID instruction. Normally the first error is the most important, with the following errors only consequences of the first error.

The first execution of a GetError or GetErrorID instruction within a block returns the first error detected during block execution. This error could have occurred anywhere between the start of the block and the execution of either GetError or GetErrorID. Subsequent executions of either GetError or GetErrorID return the first error since the previous execution of GetError or GetErrorID. The history of errors is not saved, and execution of either instruction will re-arm the PLC system to catch the next error.

The ErrorStruct data type used by the GetError instruction can be added in the data block editor and block interface editors, so your program logic can access these values. Select ErrorStruct from the data type drop-down list to add this structure. You can create multiple ErrorStruct elements by using unique names. The members of an ErrorStruct cannot be renamed.

Error condition indicated by ENO

If EN = TRUE and GetError or GetErrorID executes, then:

- ENO = TRUE indicates a code block execution error occurred and error data is present
- ENO = FALSE indicates no code block execution error occurred

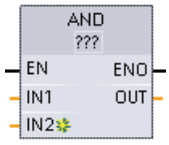
You can connect error reaction program logic to ENO which activates after an error occurs. If an error exists, then the output parameter stores the error data where your program has access to it.

GetError and GetErrorID can be used to send error information from the currently executing block (called block) to a calling block. Place the instruction in the last network of the called block program to report the final execution status of the called block.

7.9 Word logic operations

7.9.1 AND, OR, and XOR instructions

Table 7- 125 AND, OR, and XOR instruction

LAD / FBD	SCL	Description
	`out := in1 AND in2;`	AND: Logical AND
	`out := in1 OR in2;`	OR: Logical OR
	`out := in1 XOR in2;`	XOR: Logical exclusive OR

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

To add an input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

To remove an input, right-click on an input stub for one of the existing IN parameters (when there are more than the original two inputs) and select the "Delete" command.

Table 7- 126 Data types for the parameters

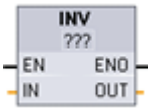
Parameter	Data type	Description
IN1, IN2	Byte, Word, DWord	Logical inputs
OUT	Byte, Word, DWord	Logical output

[1] The data type selection sets parameters IN1, IN2, and OUT to the same data type.

The corresponding bit values of IN1 and IN2 are combined to produce a binary logic result at parameter OUT. ENO is always TRUE following the execution of these instructions.

7.9.2 Invert instruction

Table 7- 127 INV instruction

LAD / FBD	SCL	Description
	`Not available`	Calculates the binary one's complement of the parameter IN. The one's complement is formed by inverting each bit value of the IN parameter (changing each 0 to 1 and each 1 to 0). ENO is always TRUE following the execution of this instruction.

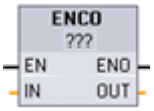
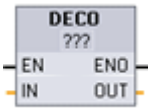
[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 128 Data types for the parameters

Parameter	Data type	Description
IN	SInt, Int, DInt, USInt, UInt, UDInt, Byte, Word, DWord	Data element to invert
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Byte, Word, DWord	Inverted output

7.9.3 Encode and decode instructions

Table 7- 129 ENCO and DECO instruction

LAD / FBD	SCL	Description
	`out := ENCO(IN := <expression>);`	Encodes a bit pattern to a binary number The ENCO instruction converts parameter IN to the binary number corresponding to the bit position of the least-significant set bit of parameter IN and returns the result to parameter OUT. If parameter IN is either 0000 0001 or 0000 0000, then a value of 0 is returned to parameter OUT. If the parameter IN value is 0000 0000, then ENO is set to FALSE.
	`out := DECO(IN := <expression>);`	Decodes a binary number to a bit pattern The DECO instruction decodes a binary number from parameter IN, by setting the corresponding bit position in parameter OUT to a 1 (all other bits are set to 0). ENO is always TRUE following execution of the DECO instruction.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 130 Data types for the parameters

Parameter	Data type	Description
IN	ENCO: Byte, Word, DWord DECO: UInt	ENCO: Bit pattern to encode DECO: Value to decode
OUT	ENCO: Int DECO: Byte, Word, DWord	ENCO: Encoded value DECO: Decoded bit pattern

Table 7- 131 OUT parameter for ENCO

ENO	Condition	Result (OUT)
1	No error	Valid bit number
0	IN is zero	OUT is set to zero

The DECO parameter OUT data type selection of a Byte, Word, or DWord restricts the useful range of parameter IN. If the value of parameter IN exceeds the useful range, then a modulo operation is performed to extract the least significant bits shown below.

DECO parameter IN range:

- 3 bits (values 0-7) IN are used to set 1 bit position in a Byte OUT
- 4-bits (values 0-15) IN are used to set 1 bit position in a Word OUT
- 5 bits (values 0-31) IN are used to set 1 bit position in a DWord OUT

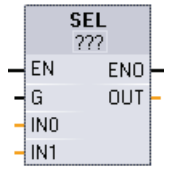
Table 7- 132 Examples

DECO IN value			DECO OUT value (Decode single bit position)
Byte OUT	Min. IN	0	00000001
8 bits	Max. IN	7	10000000
Word OUT	Min. IN	0	0000000000000001
16 bits	Max. IN	15	1000000000000000
DWord OUT	Min. IN	0	00000000000000000000000000000001
32 bits	Max. IN	31	10000000000000000000000000000000

7.9.4 Select, Multiplex, and Demultiplex instructions

Table 7- 133 SEL (select) instruction

LAD / FBD	SCL	Description

	`out := SEL(`
	`g:=_bool_in,`
	`in0:-_variant_in,`
	`in1:=_variant_in);`
		SEL assigns one of two input values to parameter OUT, depending on the parameter G value.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 134 Data types for the SEL instruction

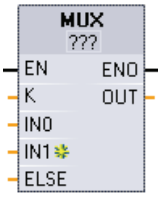
Parameter	Data type [1]	Description
G	Bool	• 0 selects IN0 • 1 selects IN1
IN0, IN1	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Output

[1] Input variables and the output variable must be of the same data type.

Condition codes: ENO is always TRUE following execution of the SEL instruction.

Table 7- 135 MUX (multiplex) instruction

LAD / FBD	SCL	Description

	`out := MUX(`
	`  k:=_unit_in,`
	`  in0:=variant_in,`
	`  in1:=variant_in,`
	
	`[...in32:=variant_in,]`
	`  inelse:=variant_in);`
		MUX copies one of many input values to parameter OUT, depending on the parameter K value. If the parameter K value exceeds (IN*n* - 1), then the parameter ELSE value is copied to parameter OUT.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

To add an input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

To remove an input, right-click on an input stub for one of the existing IN parameters (when there are more than the original two inputs) and select the "Delete" command.

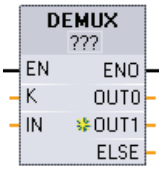
Table 7- 136 Data types for the MUX instruction

Parameter	Data type	Description
K	UInt	• 0 selects IN0 • 1 selects IN1 • *n* selects IN*n*
IN0, IN1, .. INn	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Inputs
ELSE	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Input substitute value (optional)
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Output

[1] Input variables and the output variable must be of the same data type.

Table 7- 137 DEMUX (Demultiplex) instruction

LAD / FBD	SCL	Description

	`out := DEMUX(`
	`k:=_unit_in,`
	`in:=variant_in,`
	`out0:=variant_in,`
	`out1:=variant_in,`
	
	`[...out32:=variant_in,]`
	
	`outelse:=variant_in);`
		DEMUX copies the value of the location assigned to parameter IN to one of many outputs. The value of the K parameter selects which output selected as the destination of the IN value. If the value of K is greater than the number (OUT*n* - 1) then the IN value is copied to location assigned to the ELSE parameter.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

To add an output, click the create icon or right-click on an output stub for one of the existing OUT parameters and select the "Insert output" command. To remove an output, right-click on an output stub for one of the existing OUT parameters (when there are more than the original two outputs) and select the "Delete" command.

To add an output, click the "Create" icon or right-click on an output stub for one of the existing OUT parameters and select the "Insert output" command.

To remove an output, right-click on an output stub for one of the existing OUT parameters (when there are more than the original two outputs) and select the "Delete" command.

Table 7- 138 Data types for the DEMUX instruction

Parameter	Data type [1]	Description
K	UInt	Selector value: • 0 selects OUT0 • 1 selects OUT1 • n selects OUTn
IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Input
OUT0, OUT1, .. OUTn	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Outputs
ELSE	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Substitute output when K is greater than (OUTn - 1)

[1] The input variable and the output variables must be of the same data type.

Table 7- 139 ENO status for the MUX and DEMUX instructions

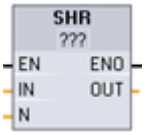
ENO	Condition	Result OUT
1	No error	MUX: Selected IN value is copied to OUT DEMUX: IN value is copied to selected OUT
0	MUX: K is greater than the number of inputs -1	• No ELSE provided: OUT is unchanged, • ELSE provided, ELSE value assigned to OUT
	DEMUX: K is greater than the number of outputs -1	• No ELSE provided: outputs are unchanged, • ELSE provided, IN value copied to ELSE

7.10 Shift and Rotate

7.10.1 Shift instructions

Table 7- 140 SHR and SHL instructions

LAD / FBD	SCL	Description

	`out := SHR(`
	`  in:=_variant_in_,`
	`  n:=_uint_in);`
	`out := SHL(`
	`  in:=_variant_in_,`
	`  n:=_uint_in);`
		Use the shift instructions (SHL and SHR) to shift the bit pattern of parameter IN. The result is assigned to parameter OUT. Parameter N specifies the number of bit positions shifted:
		• SHR: Shift bit pattern right
		• SHL: Shift bit pattern left

[1] For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

Table 7- 141 Data types for the parameters

Parameter	Data type	Description
IN	Byte, Word, DWord	Bit pattern to shift
N	UInt	Number of bit positions to shift
OUT	Byte, Word, DWord	Bit pattern after shift operation

- For N=0, no shift occurs. The IN value is assigned to OUT.
- Zeros are shifted into the bit positions emptied by the shift operation.
- If the number of positions to shift (N) exceeds the number of bits in the target value (8 for Byte, 16 for Word, 32 for DWord), then all original bit values will be shifted out and replaced with zeros (zero is assigned to OUT).
- ENO is always TRUE for the shift operations.

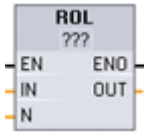
Table 7- 142 SHL example for Word data

Shift the bits of a Word to the left by inserting zeroes from the right (N = 1)			
IN	1110 0010 1010 1101	OUT value before first shift:	1110 0010 1010 1101
		After first shift left:	1100 0101 0101 1010
		After second shift left:	1000 1010 1011 0100
		After third shift left:	0001 0101 0110 1000

7.10.2 Rotate instructions

Table 7- 143 ROR and ROL instructions

LAD / FBD	SCL	Description

	`out := ROL(`
	`in:=_variant_in_,`
	`n:=_uint_in);`
	`out := ROR(`
	`in:=_variant_in_,`
	`n:=_uint_in);`
		Use the rotate instructions (ROR and ROL) to rotate the bit pattern of parameter IN. The result is assigned to parameter OUT. Parameter N defines the number of bit positions rotated.
		• ROR: Rotate bit pattern right
		• ROL: Rotate bit pattern left

[1] For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

Table 7- 144 Data types for the parameters

Parameter	Data type	Description
IN	Byte, Word, DWord	Bit pattern to rotate
N	UInt	Number of bit positions to rotate
OUT	Byte, Word, DWord	Bit pattern after rotate operation

- For N=0, no rotate occurs. The IN value is assigned to OUT.
- Bit data rotated out one side of the target value is rotated into the other side of the target value, so no original bit values are lost.
- If the number of bit positions to rotate (N) exceeds the number of bits in the target value (8 for Byte, 16 for Word, 32 for DWord), then the rotation is still performed.
- ENO is always TRUE following execution of the rotate instructions.

Table 7- 145 ROR example for Word data

Rotate bits out the right -side into the left -side (N = 1)			
IN	0100 0000 0000 0001	OUT value before first rotate:	0100 0000 0000 0001
		After first rotate right:	1010 0000 0000 0000
		After second rotate right:	0101 0000 0000 0000

Extended instructions

8

8.1 Date and time-of-day

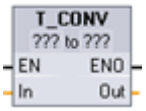
8.1.1 Date and time instructions

Use the date and time instructions to program calendar and time calculations.

- T_CONV converts the data type of a time value: (Time to DInt) or (DInt to Time)
- T_ADD adds Time and DTL values: (Time + Time = Time) or (DTL + Time = DTL)
- T_SUB subtracts Time and DTL values: (Time - Time = Time) or (DTL - Time = DTL)
- T_DIFF provides the difference between two DTL values as a Time value: DTL - DTL = Time
- T_COMBINE combines a Date value and a Time_and_Date value to create a DTL value

For information about the structure of the DTL and Time data, refer to the section on the Time and Date data types (Page 89).

Table 8- 1 T_CONV (Time Convert) instruction

LAD / FBD	SCL	Description
	`out := T_CONV( in:=_variant_in);`	T_CONV converts a Time data type to a DInt data type, or the reverse conversion from DInt data type to Time data type.

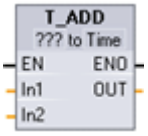
[1] For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

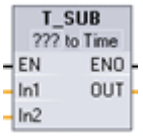
Table 8- 2 Data types for the T_CONV parameters

Parameter and type		Data type	Description
IN	IN	DInt, Time	Input Time value or DInt value
OUT	OUT	DInt, Time	Converted DInt value or Time value

Table 8- 3 T_ADD (Time Add) and T_SUB (Time Subtract) instructions

LAD / FBD	SCL	Description

	`out := T_ADD(`
	`in1:=_variant_in,`
	`in2:=_time_in);`
		T_ADD adds the input IN1 value (DTL or Time data types) with the input IN2 Time value. Parameter OUT provides the DTL or Time value result. Two data type operations are possible:
		• Time + Time = Time
		• DTL + Time = DTL

	`out := T_SUB(`
	`in1:=_variant_in,`
	`in2:=_time_in);`
		T_SUB subtracts the IN2 Time value from IN1 (DTL or Time value). Parameter OUT provides the difference value as a DTL or Time data type. Two data type operations are possible:
		• Time - Time = Time
		• DTL - Time = DTL

[1] For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

Table 8- 4 Data types for the T_ADD and T_SUB parameters

Parameter and type		Data type	Description
IN1[1]	IN	DTL, Time	DTL or Time value
IN2	IN	Time	Time value to add or subtract
OUT	OUT	DTL, Time	DTL or Time sum or difference

[1] Select the IN1 data type from the drop-down list available below the instruction name. The IN1 data type selection also sets the data type of parameter OUT.

Table 8- 5 T_DIFF (Time Difference) instruction

LAD / FBD	SCL	Description

	`out := T_DIFF(`
	`in1:=_DTL_in,`
	`in2:=_DTL_in);`
		T_DIFF subtracts the DTL value (IN2) from the DTL value (IN1). Parameter OUT provides the difference value as a Time data type.
		• DTL - DTL = Time

Table 8- 6 Data types for the T_DIFF parameters

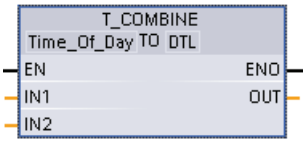
Parameter and type		Data type	Description
IN1	IN	DTL	DTL value
IN2	IN	DTL	DTL value to subtract
OUT	OUT	Time	Time difference

Condition codes: ENO = 1 means no error occurred. ENO = 0 and parameter OUT = 0 errors:

- Invalid DTL value
- Invalid Time value

Table 8- 7 T_COMBINE (combine time values) instruction

LAD / FBD	SCL	Description

	`out := CONCAT_DATE_TOD(`
	`in1 := _Time_Of_Day_in,`
	`in2 := _Time_Of_Day_in,`
	`out => _DTL_out);`
		T_COMBINE combines a Date value and a Time_of_Day value to create a DTL value.

[1] Note that the T_COMBINEinstruction in the Extended Instructions equates to the CONCAT_DATE_TODfunction in SCL.

Table 8- 8 Data types for the T_COMBINE parameters

Parameter and type		Data type	Description
IN1	IN	Date	Date value to be combined must be between DATE#1990-01-01 and DATE#2089-12-31
IN2	IN	Time_of_Day	Time_of_Day values to be combined
OUT	OUT	DTL	DTL value

8.1.2 Set and read system clock

Use the clock instructions to set and read the CPU system clock. The data type DTL (Page 89) is used to provide date and time values.

Table 8- 9 System time instructions

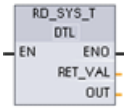
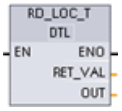
LAD / FBD	SCL	Description
	`ret_val := WR_SYS_T(` `in:=_DTL_in_);`	WR_SYS_T (Write System Time) sets the CPU time of day clock with a DTL value at parameter IN. This time value does not include local time zone or daylight saving time offsets.
	`ret_val := RD_SYS_T(` `out=>_DTL_out);`	RD_SYS_T (Read System Time) reads the current system time from the CPU. This time value does not include local time zone or daylight saving time offsets.
	`ret_val := RD_LOC_T(` `out=>_DTL_out);`	RD_LOC_T (Read Local Time) provides the current local time of the CPU as a DTL data type. This time value reflects the local time zone adjusted appropriately for daylight saving time (if configured).

Table 8- 10 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	DTL	Time of day to set in the CPU system clock
RET_VAL	OUT	Int	Execution condition code
OUT	OUT	DTL	RD_SYS_T: Current CPU system time RD_LOC_T: Current local time. including any adjustment for daylight saving time, if configured

- The local time is calculated by using the time zone and daylight saving time offsets that you set in the device configuration general tab "Time of day" parameters.
- Time zone configuration is an offset to UTC or GMT time.
- Daylight saving time configuration specifies the month, week, day, and hour when daylight saving time begins.
- Standard time configuration also specifies the month, week, day, and hour when standard time begins.
- The time zone offset is always applied to the system time value. The daylight saving time offset is only applied when daylight saving time is in effect.

Note

Daylight saving and standard start time configuration

The "Time of day" properties for "Start for daylight saving time" of the CPU device configuration must be your local time.

Condition codes: ENO = 1 means no error occurred. ENO = 0 means an execution error occurred, and a condition code is provided at the RET_VAL output.

Table 8- 11 Condition codes

RET_VAL (W#16#....)	Description
0000	The current local time is in standard time.
0001	Daylight saving time has been configured, and the current local time is in daylight saving time.
8080	Local time not available
8081	Illegal year value
8082	Illegal month value
8083	Illegal day value
8084	Illegal hour value
8085	Illegal minute value
8086	Illegal second value
8087	Illegal nanosecond value
80B0	The real-time clock has failed.

8.1.3 Run-time meter instruction

Table 8- 12 RTM instruction

LAD / FBD	SCL	Description

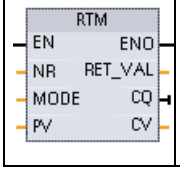
	`RTM(NR:=_uint_in_,`
	`MODE:=_byte_in_,`
	`PV:=_dint_in_,`
	`CQ=>_bool_out_,`
	`CV=>_dint_out_);`
		The RTM (Run Time Meter) instruction can set, start, stop, and read the run-time hour meters in the CPU.

Table 8- 13 Data types for the parameters

Parameter and type		Data type	Description
NR	IN	UInt	Run-time meter number: (possible values: 0..9)
MODE	IN	Byte	RTM Execution mode number: • 0 = Fetch values (the status is then written to CQ and the current value to CV) • 1 = Start (at the last counter value) • 2 = Stop • 4 = Set (to the value specified in PV) • 5 = Set (to the value specified in PV) and then start • 6 = Set (to the value specified in PV) and then stop • 7 = Save all RTM values in the CPU to the MC (Memory Card)
PV	IN	DInt	Preset hours value for the specified run-time meter
RET_VAL	OUT	Int	Function result / error message
CQ	OUT	Bool	Run-time meter status (1 = running)
CV	OUT	DInt	Current run-time hours value for the specified meter

The CPU operates up to 10 run-time hour meters to track the run-time hours of critical control subsystems. You must start the individual hour meters with one RTM execution for each timer. All run-time hour meters are stopped when the CPU makes a run-to-stop transition. You can also stop individual timers with RTM execution mode 2.

When a CPU makes a stop-to-run transition, you must restart the hour timers with one RTM execution for each timer that is started. After a run-time meter value is greater than 2147483647 hours, counting stops and the "Overflow" error is sent. You must execute the RTM instruction once for each timer to reset or modify the timer.

A CPU power failure or power cycle causes a power-down process that saves the current run-time meter values in retentive memory. Upon CPU power-up, the stored run-time meter values are reloaded to the timers and the previous run-time hour totals are not lost. The run-time meters must be restarted to accumulate additional run-time.

Your program can also use RTM execution mode 7 to save the run-time meter values in a memory card. The states of all timers at the instant RTM mode 7 is executed are stored in the memory card. These stored values can become incorrect over time as the hour timers are started and stopped during a program run session. You must periodically update the memory card values to capture important run-time events. The advantage that you get from storing the RTM values in the memory card is that you can insert the memory card in a substitute CPU where your program and saved RTM values will be available. If you did not save the RTM values in the memory card, then the timer values would be lost (in a substitute CPU).

Note

Avoid excessive program calls for memory card write operations

Minimize flash memory card write operations to extend the life of the memory card.

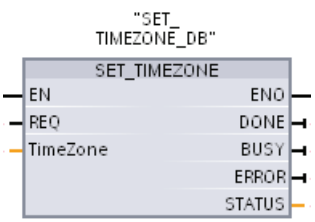
Table 8- 14 Condition codes

RET_VAL (W#16#....)	Description
0	No error
8080	Incorrect run-time meter number
8081	A negative value was passed to the parameter PV
8082	Overflow of the operating hours counter
8091	The input parameter MODE contains an illegal value.
80B1	Value cannot be saved to MC (MODE=7)

8.1.4 SET_TIMEZONE instruction

Table 8- 15 SET_TIMEZONE instruction

LAD / FBD	SCL	Description

	`"SET_TIMEZONE_DB"(`
	`Timezone:=_struct_in,`
	`DONE=>_bool_out_,`
	`BUSY=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_);`
		Sets the local time zone and daylight saving parameters that are used to transform the CPU system time to local time.

[2] In the SCL example, "SET_TIMEZONE_DB" is the name of the instance DB.

Table 8- 16 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	REQ=1: execute function
Timezone	IN	TimeTransformationRule	Rules for the transformation from system time to local time
DONE	OUT	Bool	Function complete
BUSY	OUT	Bool	Function busy
ERROR	OUT	Bool	Error detected
STATUS	OUT	Word	Function result / error message

[1] The REQ input is not used for SCL.

To manually configure the time zone parameters for the CPU, use the "Time of day" properties of the "General" tab of the device configuration.

Use the SET_TIMEZONE instruction to set the local time configuration programmatically. The parameters of the "TimeTransformationRule" structure specify the local time zone and timing for automatic switching between standard time and daylight saving time.

Table 8- 17 "TimeTransformationRule" structure

Parameter	Data type	Description
Bias	Int	Time difference between UTC and local time [min]
DaylightBias	Int	Time difference between winter and summer time [min]
DaylightStartMonth	USInt	Month of daylight saving time
DaylightStartWeek	USInt	Week of daylight saving time: • 1 = First occurrence of the weekday in the month • ... • 5 = Last occurrence of the weekday of the month
DaylightStartWeekday	USInt	Weekday of daylight saving time: • 1 = Sunday • ... • 7 = Saturday
DaylightStartHour	USInt	Hour of daylight saving time
StandardStartMonth	USInt	Month of switching to winter time
StandardStartWeek	USInt	Week of the changeover to winter time: • 1 = First occurrence of the weekday in the month • ... • 5 = last occurrence of the weekday of the month
StandardStartWeekday	USInt	Weekday of winter time: • 1 = Sunday • ... • 7 = Saturday

Parameter	Data type	Description
StandardStartHour	USInt	Hour of the winter time
Time Zone Name	STRING [80]	Name of the zone: (GMT +01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna

8.2 String and character

8.2.1 String data overview

String data type

String data is stored as a 2-byte header followed by up to 254 character bytes of ASCII character codes. A String header contains two lengths. The first byte is the maximum length that is given in square brackets when you initialize a string, or 254 by default. The second header byte is the current length that is the number of valid characters in the string. The current length must be smaller than or equal to the maximum length. The number of stored bytes occupied by the String format is 2 bytes greater than the maximum length.

Initialize your String data

String input and output data must be initialized as valid strings in memory, before execution of any string instructions.

Valid String data

A valid string has a maximum length that must be greater than zero but less than 255. The current length must be less than or equal to the maximum length.

Strings cannot be assigned to I or Q memory areas.

For more information see: Format of the String data type (Page 89).

8.2.2 S_MOVE instruction

Table 8- 18 String move instruction

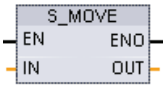
LAD / FBD	SCL	Description
	`out := in;`	Copy the source IN string to the OUT location. S_MOVE execution does not affect the contents of the source string.

Table 8- 19 Data types for the parameters

Parameter	Data type	Description
IN	String	Source string
OUT	String	Target address

If the actual length of the string at the input IN exceeds the maximum length of a string stored at output OUT, then the part of the IN string which can fit in the OUT string is copied.

8.2.3 String conversion instructions

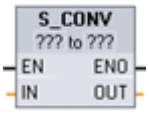
8.2.3.1 String to value and value to string conversions

You can convert number character strings to number values or number values to number character strings with these instructions:

- S_CONV converts (number string to a number value) or (number value to a number string)
- STRG_VAL converts a number string to a number value with format options
- VAL_STRG converts a number value to a number string with format options

S_CONV (String to value conversions)

Table 8- 20 String conversion instruction

LAD / FBD	SCL	Description
	`out := <Type>_TO_<Type>(in);`	Converts a character string to the corresponding value, or a value to the corresponding character string. The S_CONV instruction has no output formatting options. This makes the S_CONV instruction simpler, but less flexible than the STRG_VAL and VAL_STRG instructions.

1 For LAD / FBD: Click the "???" and select the data type from the drop-down list.

2 For SCL: Select S_CONV from the Extended Instructions, and answer the prompts for the data types for the conversion. STEP 7 then provides the appropriate conversion instruction.

Table 8- 21 Data types (string to value)

Parameter and type		Data type	Description
IN	IN	String	Input character string
OUT	OUT	String, Char, SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Output number value

Conversion of the string parameter IN starts at the first character and continues until the end of the string, or until the first character is encountered that is not "0" through "9", "+", "-", or ".". The result value is provided at the location specified in parameter OUT. If the output number value does not fit in the range of the OUT data type, then parameter OUT is set to 0 and ENO is set to FALSE. Otherwise, parameter OUT contains a valid result and ENO is set to TRUE.

Input String format rules:

- If a decimal point is used in the IN string, you must use the "." character.
- Comma characters "," used as a thousands separator to the left of the decimal point are allowed and ignored.
- Leading spaces are ignored.
- Floating-point as well as fixed-point representation is supported. The characters "e" and "E" are recognized as exponential notation.

S_CONV (Value to string conversions)

Table 8- 22 Data types (value to string)

Parameter and type		Data type	Description
IN	IN	String, Char, SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input number value
OUT	OUT	String	Output character string

An integer, unsigned integer, or floating point value IN is converted to the corresponding character string at OUT. The parameter OUT must reference a valid string before the conversion is executed. A valid string consists of a maximum string length in the first byte, the current string length in the second byte, and the current string characters in the next bytes. The converted string replaces characters in the OUT string starting at the first character and adjusts the current length byte of the OUT string. The maximum length byte of the OUT string is not changed.

How many characters are replaced depends on the parameter IN data type and number value. The number of characters replaced must fit within the parameter OUT string length. The maximum string length (first byte) of the OUT string should be greater than or equal to the maximum expected number of converted characters. The following table shows the maximum possible string lengths required for each supported data type.

Table 8- 23 Maximum string lengths for each data type

IN data type	Maximum number of converted characters in OUT string	Example	Total string length including maximum and current length bytes
USInt	3	255	5
SInt	4	-128	6
UInt	5	65535	7
Int	6	-32768	8
UDInt	10	4294967295	12
DInt	11	-2147483648	13

Output String format rules:

- Values written to parameter OUT do not use a leading "+" sign.
- Fixed-point representation is used (no exponential notation).
- The period character "." is used to represent the decimal point when parameter IN is the Real data type.

STRG_VAL instruction

Table 8- 24 String-to-value instruction

LAD / FBD	SCL	Description

	`"STRG_VAL"(`
	`in:=_string_in,`
	`format:=_word_in,`
	`p:=uint_in,`
	`out=>_variant_out);`
		Converts a number character string to the corresponding integer or floating point representation.

[1] For LAD / FBD: Click the "???" and select the data type from the drop-down list.

Table 8- 25 Data types for the STRG_VAL instruction

Parameter and type		Data type	Description
IN	IN	String	The ASCII character string to convert
FORMAT	IN	Word	Output format options
P	IN	UInt, Byte, USInt	IN: Index to the first character to be converted (first character = 1)
OUT	OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Converted number value

Conversion begins in the string IN at character offset P and continues until the end of the string, or until the first character is encountered that is not "+", "-", ".", ",", "e", "E", or "0" to "9". The result is placed at the location specified in parameter OUT.

String data must be initialized before execution as a valid string in memory.

The FORMAT parameter for the STRG_VAL instruction is defined below. The unused bit positions must be set to zero.

Table 8- 26 Format of the STRG_VAL instruction

Bit 16							Bit 8	Bit 7							Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	f	r

f = Notation format
1= Exponential notation
0 = Fixed point notation

r = Decimal point format
1 = "," (comma character)
0 = "." (period character)

Table 8- 27 Values of the FORMAT parameter

FORMAT (W#16#)	Notation format	Decimal point representation
0000 (default)	Fixed point	"."
0001		","
0002	Exponential	"."
0003		","
0004 to FFFF	Illegal values	

Rules for STRG_VAL conversion:

- If the period character "." is used for the decimal point, then commas "," to the left of the decimal point are interpreted as thousands separator characters. The comma characters are allowed and ignored.
- If the comma character "," is used for the decimal point, then periods "." to the left of the decimal point are interpreted as thousands separator characters. These period characters are allowed and ignored.
- Leading spaces are ignored.

VAL_STRG instruction

Table 8- 28 Value-to-string operation

LAD / FBD	SCL	Description

	`"VAL_STRG"(`
	`in:=_variant_in,`
	`size:=_usint_in,`
	`prec:=_usint_in,`
	`format:=_word_in,`
	`p:=uint_in,`
	`out=>_string_out);`
		Converts an integer, unsigned integer, or floating point value to the corresponding character string representation.

[1] For LAD / FBD: Click the "???" and select the data type from the drop-down list.

Table 8- 29 Data types for the VAL_STRG instruction

Parameter and type		Data type	Description
IN	IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Value to convert
SIZE	IN	USInt	Number of characters to be written to the OUT string
PREC	IN	USInt	The precision or size of the fractional portion. This does not include the decimal point.
FORMAT	IN	Word	Output format options
P	IN	UInt, Byte, USInt	IN: Index to the first OUT string character to be replaced (first character = 1)
OUT	OUT	String	The converted string

The value represented by parameter IN is converted to a string referenced by parameter OUT. The parameter OUT must be a valid string before the conversion is executed.

The converted string will replace characters in the OUT string starting at character offset count P to the number of characters specified by parameter SIZE. The number of characters in SIZE must fit within the OUT string length, counting from character position P. This instruction is useful for embedding number characters into a text string. For example, you can put the numbers "120" into the string "Pump pressure = 120 psi".

Parameter PREC specifies the precision or number of digits for the fractional part of the string. If the parameter IN value is an integer, then PREC specifies the location of the decimal point. For example, if the data value is 123 and PREC = 1, then the result is "12.3". The maximum supported precision for the Real data type is 7 digits.

If parameter P is greater than the current size of the OUT string, then spaces are added, up to position P, and the result is appended to the end of the string. The conversion ends if the maximum OUT string length is reached.

The FORMAT parameter for the VAL_STRG instruction is defined below. The unused bit positions must be set to zero.

Table 8- 30 Format of the VAL_STRG instruction

Bit 16							Bit 8	Bit 7							Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	s	f	r

s = Number sign character
1= use sign character "+" and "-"
0 = use sign character "-" only

f = Notation format
1= Exponential notation
0 = Fixed point notation

r = Decimal point format
1 = "," (comma character)
0 = "." (period character)

Table 8- 31 Values of the FORMAT parameter

FORMAT (WORD)	Number sign character	Notation format	Decimal point representation
W#16#0000	"-" only	Fixed point	"."
W#16#0001			","
W#16#0002		Exponential	"."
W#16#0003			","
W#16#0004	"+" and "-"	Fixed Point	"."
W#16#0005			","
W#16#0006		Exponential	"."
W#16#0007			","
W#16#0008 to W#16#FFFF	Illegal values		

Parameter OUT string format rules:

- Leading space characters are added to the leftmost part of the string when the converted string is smaller than the specified size.
- When the FORMAT parameter sign bit is FALSE, unsigned and signed integer data type values are written to the output buffer without the leading "+" sign. The "-" sign is used if required.
  <leading spaces><digits without leading zeroes>'.'<PREC digits>
- When the sign bit is TRUE, unsigned and signed integer data type values are written to the output buffer always with a leading sign character.

  <leading spaces><sign><digits without leading zeroes>'.'<PREC digits>
- When the FORMAT is set to exponential notation, Real data type values are written to the output buffer as:

  <leading spaces><sign><digit> '.' <PREC digits>'E' <sign><digits without leading zero>
- When the FORMAT is set to fixed point notation, integer, unsigned integer, and real data type values are written to the output buffer as:

  <leading spaces><sign><digits without leading zeroes>'.'<PREC digits>
- Leading zeros to the left of the decimal point (except the digit adjacent to the decimal point) are suppressed.
- Values to the right of the decimal point are rounded to fit in the number of digits to the right of the decimal point specified by the PREC parameter.
- The size of the output string must be a minimum of three bytes more than the number of digits to the right of the decimal point.
- Values are right-justified in the output string.

Conditions reported by ENO

When an error is encountered during the conversion operation, the following results will be returned:

- ENO is set to 0.
- OUT is set to 0, or as shown in the examples for string to value conversion.
- OUT is unchanged, or as shown in the examples when OUT is a string.

Table 8- 32 ENO status

ENO	Description
1	No error
0	Illegal or invalid parameter; for example, an access to a DB that does not exist
0	Illegal string where the maximum length of the string is 0 or 255
0	Illegal string where the current length is greater than the maximum length
0	The converted number value is too large for the specified OUT data type.
0	The OUT parameter maximum string size must be large enough to accept the number of characters specified by parameter SIZE, starting at the character position parameter P.
0	Illegal P value where P=0 or P is greater than the current string length
0	Parameter SIZE must be greater than parameter PREC.

Table 8- 33 Examples of S_CONV string to value conversion

IN string	OUT data type	OUT value	ENO
"123"	Int or DInt	123	TRUE
"-00456"	Int or DInt	-456	TRUE
"123.45"	Int or DInt	123	TRUE
"+2345"	Int or DInt	2345	TRUE
"00123AB"	Int or DInt	123	TRUE
"123"	Real	123.0	TRUE
"123.45"	Real	123.45	TRUE
"1.23e-4"	Real	1.23	TRUE
"1.23E-4"	Real	1.23	TRUE
"12,345.67"	Real	12345.67	TRUE
"3.4e39"	Real	3.4	TRUE
"-3.4e39"	Real	-3.4	TRUE
"1.17549e-38"	Real	1.17549	TRUE
"12345"	SInt	0	FALSE
"A123"	N/A	0	FALSE
""	N/A	0	FALSE
"++123"	N/A	0	FALSE
"+-123"	N/A	0	FALSE

Table 8- 34 Examples of S_CONV value to string conversion

Data type	IN value	OUT string	ENO
UInt	123	"123"	TRUE
UInt	0	"0"	TRUE
UDInt	12345678	"12345678"	TRUE
Real	-INF	"INF"	FALSE
Real	+INF	"INF"	FALSE
Real	NaN	"NaN"	FALSE

Table 8- 35 Examples of STRG_VAL conversion

IN string	FORMAT (W#16#....)	OUT data type	OUT value	ENO
"123"	0000	Int or DInt	123	TRUE
"-00456"	0000	Int or DInt	-456	TRUE
"123.45"	0000	Int or DInt	123	TRUE
"+2345"	0000	Int or DInt	2345	TRUE
"00123AB"	0000	Int or DInt	123	TRUE
"123"	0000	Real	123.0	TRUE
"-00456"	0001	Real	-456.0	TRUE
"+00456"	0001	Real	456.0	TRUE
"123.45"	0000	Real	123.45	TRUE
"123.45"	0001	Real	12345.0	TRUE
"123,45"	0000	Real	12345.0	TRUE
"123,45"	0001	Real	123.45	TRUE
".00123AB"	0001	Real	123.0	TRUE
"1.23e-4"	0000	Real	1.23	TRUE
"1.23E-4"	0000	Real	1.23	TRUE
"1.23E-4"	0002	Real	1.23E-4	TRUE
"12,345.67"	0000	Real	12345.67	TRUE
"12,345.67"	0001	Real	12.345	TRUE
"3.4e39"	0002	Real	+INF	TRUE
"-3.4e39"	0002	Real	-INF	TRUE
"1.1754943e-38" (and smaller)	0002	Real	0.0	TRUE
"12345"	N/A	SInt	0	FALSE
"A123"	N/A	N/A	0	FALSE
""	N/A	N/A	0	FALSE
"++123"	N/A	N/A	0	FALSE
"+-123"	N/A	N/A	0	FALSE

The following examples of VAL_STRG conversions are based on an OUT string initialized as follows:

"Current Temp = xxxxxxxxxx C"
where the "x" character represents space characters allocated for the converted value.

Table 8- 36 Examples of VAL_STRG conversion

Data type	IN value	P	SIZE	FORMAT (W#16#....)	PREC	OUT string	ENO
UInt	123	16	10	0000	0	Current Temp = xxxxxxx123 C	TRUE
UInt	0	16	10	0000	2	Current Temp = xxxxxx0.00 C	TRUE
UDInt	12345678	16	10	0000	3	Current Temp = x12345.678 C	TRUE
UDInt	12345678	16	10	0001	3	Current Temp = x12345,678 C	TRUE
Int	123	16	10	0004	0	Current Temp = xxxxxx+123 C	TRUE
Int	-123	16	10	0004	0	Current Temp = xxxxxx-123 C	TRUE
Real	-0.00123	16	10	0004	4	Current Temp = xxx-0.0012 C	TRUE
Real	-0.00123	16	10	0006	4	Current Temp = -1.2300E-3 C	TRUE
Real	-INF	16	10	N/A	4	Current Temp = xxxxxx-INF C	FALSE
Real	+INF	16	10	N/A	4	Current Temp = xxxxxx+INF C	FALSE
Real	NaN	16	10	N/A	4	Current Temp = xxxxxxxNaN C	FALSE
UDInt	12345678	16	6	N/A	3	Current Temp = xxxxxxxxxx C	FALSE

8.2.3.2 String-to-characters and characters-to-string conversions

Chars_TO_Strg copies an array of ASCII character bytes into a character string.

Strg_TO_Chars copies an ASCII character string into an array of character bytes.

Note

Only the zero based array types (Array [0..n] of Char) or (Array [0..n] of Byte) are allowed as the input parameter Chars for the Chars_TO_Strg instruction, or as the IN_OUT parameter Chars for the Strg_TO_Chars instruction.

Table 8- 37 Chars_TO_Strg instruction

LAD / FBD	SCL	Description

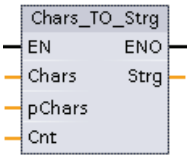
	`Chars_TO_Strg(`
	`Chars:=_variant_in_,`
	`pChars:=_dint_in_,`
	`Cnt:=_uint_in_,`
	`Strg=>_string_out_);`
		All or part of an array of characters is copied to a string.
		The output string must be declared before Chars_TO_Strg is executed. The string is then overwritten by the Chars_TO_Strg operation.
		Strings of all supported maximum lengths (1..254) may be used.
		The string maximum length value is not changed by Chars_TO_Strg operation. Copying from array to string stops when the maximum string length is reached.
		A nul character '$00' or 16#00 value in the character array works as a delimiter and ends copying of characters into the string.

Table 8- 38 Data types for the parameters (Chars_TO_Strg)

Parameter and type		Data type	Description
Chars	IN	Variant	The Chars parameter is a pointer to zero based array [0..n] of characters to be converted into a string. The array can be declared in a DB or as local variables in the block interface. Example: "DB1".MyArray points to MyArray [0..10] of Char element values in DB1.
pChars	IN	Dint	Element number for the first character in the array to copy. Array element [0] is the default value.
Cnt	IN	UInt	Count of characters to copy: 0 means all
Strg	OUT	String	Target string

Table 8- 39 Strg_TO_Chars instruction

LAD / FBD	SCL	Description

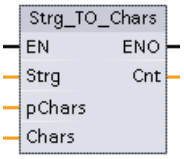
	`Strg_TO_Chars(`
	`Strg:=_string_in_,`
	`pChars:=_dint_in_,`
	`Cnt=>_uint_out_,`
	`Chars:=_variant_inout_);`
		The complete input string Strg is copied to an array of characters at IN_OUT parameter Chars.
		The operation overwrites bytes starting at array element number specified by the pChars parameter.
		Strings of all supported max lengths (1..254) may be used.
		An end delimiter is not written; this is your responsibility. To set an end delimiter just after the last written array character, use the next array element number [pChars+Cnt].

Table 8- 40 Data types for the parameters (Strg_TO_Chars)

Parameter and type		Data type	Description
Strg	IN	String	Source string
pChars	IN	DInt	Array element number for the first string character written to the target array

Parameter and type		Data type	Description
Chars	IN_OUT	Variant	The Chars parameter is a pointer to zero based array [0..n] of characters copied from the input string. The array can be declared in a DB or as local variables in the block interface. Example: "DB1".MyArray points to MyArray [0..10] of Char element values in DB1.
Cnt	OUT	UInt	Count of characters copied

Table 8- 41 ENO status

ENO	Description
1	No error
0	Chars_TO_Strg: Attempt to copy more character bytes to the output string than allowed by the maximum length byte in the string declaration
0	Chars_TO_Strg: The nul character (16#00) value was found in the input character byte array.
0	Strg_TO_Chars: Attempt to copy more character bytes to the output array than are allowed by the element number limit

8.2.3.3 ASCII to Hex and Hex to ASCII conversions

Use the ATH (ASCII to hexadecimal) and HTA (hexadecimal to ASCII) instructions for conversions between ASCII character bytes (characters 0 to 9 and uppercase A to F only) and the corresponding 4-bit hexadecimal nibbles.

Table 8- 42 ATH instruction

LAD / FBD	SCL	Description

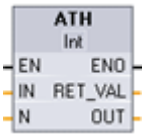
	`ret_val := ATH(`
	`in:=_variant_in_,`
	`n:=_int_in_,`
	`out=>_variant_out_);`
		Converts ASCII characters into packed hexadecimal digits.

Table 8- 43 Data types for the ATH instruction

Parameter type		Data Type	Description
IN	IN	Variant	Pointer to ASCII character byte array
N	IN	UInt	Number of ASCII character bytes to convert
RET_VAL	OUT	Word	Execution condition code
OUT	OUT	Variant	Pointer to the converted hexadecimal byte array

Conversion begins at the location specified by parameter IN and continues for N bytes. The result is placed at the location specified by OUT. Only valid ASCII characters 0 to 9 and uppercase A to F can be converted. Any other character will be converted to zero.

8-bit ASCII coded characters are converted to 4-bit hexadecimal nibbles. Two ASCII characters can be stored in a single byte.

The IN and OUT parameters specify byte arrays and not hexadecimal String data. ASCII characters are converted and placed in the hexadecimal output in the same order as they are read. If there are an odd number of ASCII characters, then zeros are put in the right-most nibble of the last converted hexadecimal digit.

Table 8- 44 ATH condition codes

RET_VAL (W#16#....)	Description	ENO
0000	No error	TRUE
0007	Invalid ATH input character	FALSE

Table 8- 45 Examples of ASCII-to-hexadecimal (ATH) conversion

IN character bytes	N	OUT value	ENO
'0123'	4	W#16#0123	TRUE
'123AFx1a23'	10	DW#16#123AF01023	FALSE
'a23'	3	W#16#A230	TRUE

Table 8- 46 HTA instruction

LAD / FBD	SCL	Description

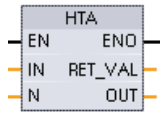
	`ret_val := HTA(`
	`  in:=_variant_in_,`
	`  n:=_uint_in_,`
	`  out=>_variant_out_);`
		Converts packed hexadecimal digits to their corresponding ASCII character bytes.

Table 8- 47 Data types for the HTA instruction

Parameter and type		Data Type	Description
IN	IN	Variant	Pointer to input byte array
N	IN	UInt	Number of bytes to convert (each input byte has two 4-bit nibbles and produces 2N ASCII characters)
RET_VAL	OUT	Word	Execution condition code
OUT	OUT	Variant	Pointer to ASCII character byte array

Conversion begins at the location specified by parameter IN and continues for N bytes. Each 4-bit nibble converts to a single 8-bit ASCII character and produces 2N ASCII character bytes of output. All 2N bytes of the output are written as ASCII characters 0 to 9 through uppercase A to F. The parameter OUT specifies a byte array and not a string.

Each nibble of the hexadecimal byte is converted into a character in the same order as they are read in (left-most nibble of a hexadecimal digit is converted first, followed by the right-most nibble of that same byte).

Table 8- 48 Examples of hexadecimal -to- ASCII (HTA) conversion

IN value	N	OUT character bytes	ENO (ENO always TRUE after HTA execution)
W#16#0123	2	'0123'	TRUE
DW#16#123AF012	4	'123AF012'	TRUE

Table 8- 49 ATH and HTA condition codes

RET_VAL (W#16#....)	Description	ENO
0000	No error	TRUE
0007	Invalid ATH input character: A character was found that was not an ASCII character 0-9, lowercase a-f, or uppercase A-F	FALSE
8101	Illegal or invalid input pointer, for example, an access to a DB that does not exist.	FALSE
8120	Input string is an invalid format, i.e., max= 0, max=255, current>max, or grant length in pointer < max	FALSE
8182	Input buffer is too small for N	FALSE
8151	Data type not allowed for input buffer	FALSE
8301	Illegal or invalid output pointer, for example, an access to a DB that does not exist.	FALSE
8320	Output string is an invalid format, i.e., max= 0, max=255, current>max, or grant length in pointer < max	FALSE
8382	Output buffer is too small for N	FALSE
8351	Data type not allowed for output buffer	FALSE

8.2.4 String operation instructions

Your control program can use the following string and character instructions to create messages for operator display and process logs.

8.2.4.1 LEN

Table 8- 50 Length instruction

LAD / FBD	SCL	Description
	`out := LEN(in);`	LEN (length) provides the current length of the string IN at output OUT. An empty string has a length of zero.

Table 8- 51 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	String	Input string
OUT	OUT	Int, DInt, Real, LReal	Number of valid characters of IN string

Table 8- 52 ENO status

ENO	Condition	OUT
1	No invalid string condition	Valid string length
0	Current length of IN exceeds maximum length of IN	Current length is set to 0
	Maximum length of IN does not fit within allocated memory range	
	Maximum length of IN is 255 (illegal length)	

8.2.4.2 CONCAT

Table 8- 53 Concatenate strings instruction

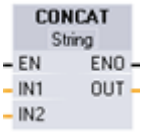
LAD / FBD	SCL	Description
	`out := CONCAT(in1, in2);`	CONCAT (concatenate strings) joins string parameters IN1 and IN2 to form one string provided at OUT. After concatenation, String IN1 is the left part and String IN2 is the right part of the combined string.

Table 8- 54 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Input string 1
IN2	IN	String	Input string 2
OUT	OUT	String	Combined string (string 1 + string 2)

Table 8- 55 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	Resulting string after concatenation is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of the OUT is reached
	Current length of IN1 exceeds maximum length of IN1, current length of IN2 exceeds maximum length of IN2, or current length of OUT exceeds maximum length of OUT (invalid string)	Current length is set to 0
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	

ENO	Condition	OUT
	Maximum length of IN1 or IN2 is 255, or the maximum length of OUT is 0 or 255	

8.2.4.3 LEFT, RIGHT, and MID

Table 8- 56 Left, right and middle substring operations

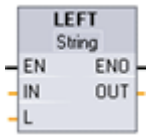
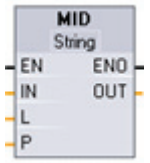
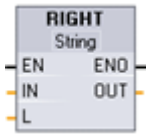
LAD / FBD	SCL	Description
	`out := LEFT(in, L);`	LEFT (Left substring) provides a substring made of the first L characters of string parameter IN. • If L is greater than the current length of the IN string, then the entire IN string is returned in OUT. • If an empty string is the input, then an empty string is returned in OUT.
	`out := MID(in, L, p);`	MID (Middle substring) provides the middle part of a string. The middle substring is L characters long and starts at character position P (inclusive). If the sum of L and P exceeds the current length of the string parameter IN, then a substring is returned that starts at character position P and continues to the end of the IN string.
	`out := RIGHT(in, L);`	RIGHT (Right substring) provides the last L characters of a string. • If L is greater than the current length of the IN string, then the entire IN string is returned in parameter OUT. • If an empty string is the input, then an empty string is returned in OUT.

Table 8- 57 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	String	Input string
L	IN	Int	Length of the substring to be created: • LEFT uses the left-most characters number of characters in the string • RIGHT uses the right-most number of characters in the string • MID uses the number of characters starting at position P within the string
P	IN	Int	MID only: Position of first substring character to be copied P= 1, for the initial character position of the IN string
OUT	OUT	String	Output string

Table 8- 58 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	• L or P is less than or equal to 0 • P is greater than maximum length of IN • Current length of IN exceeds maximum length of IN, or current length of OUT exceeds maximum length of OUT • Maximum length of IN or OUT does not fit within allocated memory • Maximum length of IN or OUT is 0 or 255	Current length is set to 0
	Substring length (L) to be copied is larger than maximum length of OUT string.	Characters are copied until the maximum length of OUT is reached
	MID only: L or P is less than or equal to 0	Current length is set to 0
	MID only: P is greater than maximum length of IN	
	Current length of IN1 exceeds maximum length of IN1, or current length of IN2 exceeds maximum length of IN2 (invalid string)	Current length is set to 0
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	
	Maximum length of IN1, IN2 or OUT is 0 or 255 (illegal length)	

8.2.4.4 DELETE

Table 8- 59 Delete substring instruction

LAD / FBD	SCL	Description
	`out := DELETE(in, L, p);`	Deletes L characters from string IN. Character deletion starts at character position P (inclusive), and the remaining substring is provided at parameter OUT. • If L is equal to zero, then the input string is returned in OUT. • If the sum of L and P is greater than the length of the input string, then the string is deleted to the end.

Table 8- 60 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	String	Input string
L	IN	Int	Number of characters to be deleted
P	IN	Int	Position of the first character to be deleted: The first character of the IN string is position number 1
OUT	OUT	String	Output string

Table 8- 61 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	P is greater than current length of IN	IN is copied to OUT with no characters deleted
	Resulting string after characters are deleted is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of OUT is reached
	L is less than 0, or P is less than or equal to 0	Current length is set to 0
	Current length of IN exceeds maximum length of IN, or current length of OUT exceeds maximum length of OUT	
	Maximum length of IN or OUT does not fit within allocated memory	
	Maximum length of IN or OUT is 0 or 255	

8.2.4.5 INSERT

Table 8- 62 Insert substring instruction

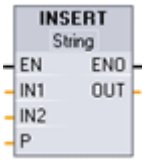
LAD / FBD	SCL	Description
	`out := INSERT(in1, in2, p);`	Inserts string IN2 into string IN1. Insertion begins after the character at position P.

Table 8- 63 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Input string 1
IN2	IN	String	Input string 2
P	IN	Int	Last character position in string IN1 before the insertion point for string IN2 The first character of string IN1 is position number 1.
OUT	OUT	String	Result string

Table 8- 64 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	P is greater than length of IN1	IN2 is concatenated with IN1 immediately following the last IN1 character
	P is less than 0	Current length is set to 0

ENO	Condition	OUT
	Resulting string after insertion is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of OUT is reached
	Current length of IN1 exceeds maximum length of IN1, current length of IN2 exceeds maximum length of IN2, or current length of OUT exceeds maximum length of OUT (invalid string)	Current length is set to 0
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	
	Maximum length of IN1 or IN2 is 255, or maximum length of OUT is 0 or 255	

8.2.4.6 REPLACE

Table 8- 65 Replace substring instruction

LAD / FBD	SCL	Description

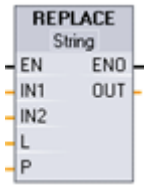
	`out := REPLACE(`
	`in1:=_string_in_,`
	`in2:=_string_in_,`
	`L:=_int_in_,`
	`p:=_int_in);`
		Replaces L characters in the string parameter IN1. Replacement starts at string IN1 character position P (inclusive), with replacement characters coming from the string parameter IN2.

Table 8- 66 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Input string
IN2	IN	String	String of replacement characters
L	IN	Int	Number of characters to replace
P	IN	Int	Position of first character to be replaced
OUT	OUT	String	Result string

If parameter L is equal to zero, then the string IN2 is inserted at position P of string IN1 without deleting any characters from string IN1.

If P is equal to one, then the first L characters of string IN1 are replaced with string IN2 characters.

Table 8- 67 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	P is greater than length of IN1	IN2 is concatenated with IN1 immediately following the last IN1 character
	P points within IN1, but fewer than L characters remain in IN1	IN2 replaces the end characters of IN1 beginning at position P
	Resulting string after replacement is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of OUT is reached
	Maximum length of IN1 is 0	IN2 characters are copied to OUT
	L is less than 0, or P is less than or equal to 0	Current length is set to 0
	Current length of IN1 exceeds maximum length of IN1, current length of IN2 exceeds maximum length of IN2, or current length of OUT exceeds maximum length of OUT	
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	
	Maximum length of IN1 or IN2 is 255, or maximum length of OUT is 0 or 255	

8.2.4.7 FIND

Table 8- 68 Find substring instruction

LAD / FBD	SCL	Description

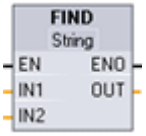
	`out := FIND(`
	`in1:=_string_in_,`
	`in2:=_string_in);`
		Provides the character position of the substring specified by IN2 within the string IN1. The search starts on the left. The character position of the first occurrence of IN2 string is returned at OUT. If the string IN2 is not found in the string IN1, then zero is returned.

Table 8- 69 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Search inside this string
IN2	IN	String	Search for this string
OUT	OUT	Int	Character position in string IN1 of the first search match

Table 8- 70 ENO status

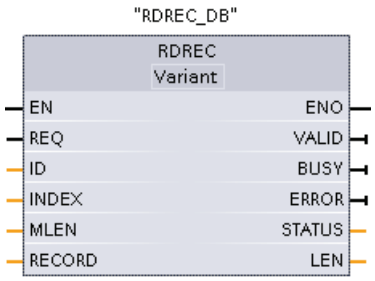
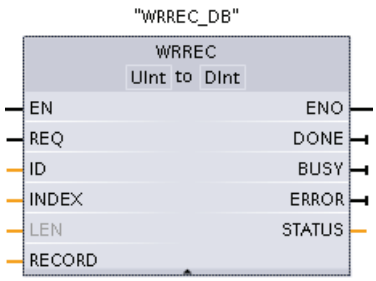
ENO	Condition	OUT
1	No errors detected	Valid character position
0	IN2 is larger than IN1	Character position is set to 0
	Current length of IN1 exceeds maximum length of IN1, or current length of IN2 exceeds maximum length of IN2 (invalid string)	
	Maximum length of IN1 or IN2 does not fit within allocated memory range	
	Maximum length of IN1 or IN2 is 255	

8.3 Distributed I/O (PROFINET, PROFIBUS, or AS-i)

8.3.1 RDREC and WRREC

You can use the RDREC (Read record) and WRREC (Write record) instructions with PROFINET, PROFIBUS, and AS-i.

Table 8- 71 RDREC and WRREC instructions

LAD / FBD	SCL	Description
	"RDREC_DB"(req:=_bool_in_, ID:=_word_in_, index:=_dint_in_, mlen:=_uint_in_, valid=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_dword_out_, len=>_uint_out_, record:=_variant_inout_);	Use the RDREC instruction to read a data record with the number INDEX from the component addressed by the ID, such as a central rack or a distributed component (PROFIBUS DP or PROFINET IO). Assign the maximum number of bytes to read in MLEN. The selected length of the target area RECORD should have at least the length of MLEN bytes.
	"WRREC_DB"(req:=_bool_in_, ID:=_word_in_, index:=_dint_in_, len:=_uint_in_, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_dword_out_, record:=_variant_inout_);	Use the WRREC instruction to transfer a data RECORD with the record number INDEX to a DP slave/PROFINET IO device component addressed by ID, such as a module in the central rack or a distributed component (PROFIBUS DP or PROFINET IO). Assign the byte length of the data record to be transmitted. The selected length of the source area RECORD should, therefore, have at least the length of LEN bytes.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL examples, "RDREC_DB" and "WRREC_DB" are the names of the instance DBs.

Table 8- 72 RDREC and WRREC data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	REQ = 1: Transfer data record
ID	IN	HW_IO (Word)	Logical address of the DP slave/PROFINET IO component (module or submodule): • For an output module, bit 15 must be set (for example, for address 5: ID:= DW#16#8005). • For a combination module, the smaller of the two addresses should be specified. **Note**: The device ID can be determined in one of two ways: • By making the following "Network view" selections: – Device (gray box) – "Properties" of the device – "Hardware identifier" **Note**: Not all devices display their Hardware identifiers, however. • By making the following "Project tree" menu selections: – PLC tags – Default tag table – System constants tab All configured device Hardware identifiers are displayed.
INDEX	IN	Byte, Word, USInt, UInt, SInt, Int, DInt	Data record number
MLEN	IN	Byte, USInt, UInt	Maximum length in bytes of the data record information to be fetched (RDREC)
VALID	OUT	Bool	New data record was received and valid (RDREC). The VALID bit is TRUE for one scan, after the last request was completed with no error.
DONE	OUT	Bool	Data record was transferred (WRREC). The DONE bit is TRUE for one scan, after the last request was completed with no error.
BUSY	OUT	Bool	• BUSY = 1: The read (RDREC) or write (WRREC) process is not yet terminated. • BUSY = 0: Data record transmission is completed.
ERROR	OUT	Bool	ERROR = 1: A read (RDREC) or write (WRREC) error has occurred. The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	DWord	Block status or error information

Parameter and type		Data type	Description
LEN	OUT (RDREC) IN (WRREC)	UInt	• Length of the fetched data record information (RDREC) • Maximum byte length of the data record to be transferred (WRREC)
RECORD	IN_OUT	Variant	• Target area for the fetched data record (RDREC) • Data record (WRREC)

The RDREC and WRREC instructions operate asynchronously, that is, processing covers multiple instruction calls. Start the job by calling RDREC or WRREC with REQ = 1.

The job status is displayed via output parameter BUSY and the two central bytes of output parameter STATUS. The transfer of the data record is complete when the output parameter BUSY has the value FALSE

TRUE (only for one scan) on the output parameter VALID (RDREC) or DONE (WRREC) verifies that the data record has been successfully transferred into the target area RECORD (RDREC) or to the target device (WRREC). In the case of the RDREC, the output parameter LEN contains the length of the fetched data in bytes.

The output parameter ERROR (only for one scan when ERROR = TRUE) indicates that a data record transmission error has occurred. In this case, the output parameter STATUS (only for the one scan when ERROR = TRUE) contains the error information.

Data records are defined by the hardware device manufacturer. Refer to the hardware manufacturer's device documentation for details about a data record.

Note

If a DPV1 slave is configured via GSD file (GSD rev. 3 and higher) and the DP interface of the DP master is set to "S7 compatible", then you may not read any data records from the I/O modules in the user program with "RDREC" or write to the I/O modules with "WRREC". In this case, the DP master addresses the wrong slot (configured slot + 3).

Remedy: set the interface of the DP master to "DPV1".

Note

The interfaces of the "RDREC" and "WRREC" instructions are identical to the "RDREC" and "WRREC" FBs defined in "PROFIBUS Guideline PROFIBUS Communication and Proxy Function Blocks according to IEC 61131-3".

Note

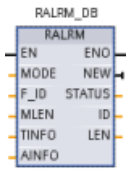
If you use "RDREC" or "WRREC" to read or write a data record for PROFINET IO, then negative values in the INDEX, MLEN, and LEN parameters will be interpreted as an unsigned 16-bit integer.

8.3.2 RALRM

You can use the RALRM (Read alarm) instruction with PROFINET and PROFIBUS.

Table 8- 73 RALRM instruction

LAD / FBD	SCL	Description

	`"RALRM_DB"(`
	`mode:=_int_in_,`
	`f_ID:=_word_in_,`
	`mlen:=_uint_in_,`
	`new=>_bool_out_,`
	`status=>_dword_out_,`
	`ID=>_word_out_,`
	`len=>_uint_out_,`
	`tinfo:=_variant_inout_,`
	`ainfo:=_variant_inout_);`
		Use the RALRM (read alarm) instruction to read diagnostic interrupt information from a DP slave or PROFINET I/O device.
		The information in the output parameters contains the start information of the called OB as well as information of the interrupt source.
		Call RALRM only within the interrupt OB that was started by the CPU operating system as a response to the peripheral I/O interrupt that you want to examine.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "RALRM_DB" is the name of the instance DB.

Table 8- 74 Data types for the parameters

Parameter and type		Data type	Description
MODE	IN	Byte, USInt, SInt, Int	Operating mode
F_ID	IN	HW_IO (Word)	Logical start address of the component (module) from which interrupts are to be received **Note**: The device ID can be determined in one of two ways: • By making the following "Network view" selections: – Device (gray box) – "Properties" of the device – "Hardware identifier" **Note**: Not all devices display their Hardware identifiers, however. • By making the following "Project tree" menu selections: – PLC tags – Default tag table – System constants tab – All configured device Hardware identifiers are displayed.
MLEN	IN	Byte, USInt, UInt	Maximum length in bytes of the data interrupt information to be received
NEW	OUT	Bool	A new interrupt was received.
STATUS	OUT	DWord	Error code of the DP Master

Parameter and type		Data type	Description
ID	OUT	HW_IO (Word)	Logical start address of the component (module) from which an interrupt was received. Bit 15 contains the I/O ID: • 0 for an input address • 1 for and output address **Note**: Refer to the F_ID parameter for an explanation of how to determine the device ID.
LEN	OUT	DWord, UInt, UDInt, DInt, Real, LReal	Length of the received interrupt information
TINFO	IN_OUT	Variant	Task information: Target range for OB start and management information
AINFO	IN_OUT	Variant	Interrupt information: Target area for header information and additional interrupt information. For AINFO, provide a length of at least the MLEN bytes.

Note

If you call "RALRM" in an OB whose start event is not an I/O interrupt, the instruction will provide correspondingly reduced information in its outputs.

Make sure to use different instance DBs when you call "RALRM" in different OBs. If you evaluate data resulting from a "RALRM" call outside of the associated interrupt OB, you should use a separate instance DB per OB start event.

Note

The interface of the "RALRM" instruction is identical to the "RALRM" FB defined in "PROFIBUS Guideline PROFIBUS Communication and Proxy Function Blocks according to IEC 61131-3".

Calling RALRM

You can call the RALRM instruction in three different operating modes (MODE).

Table 8- 75 RALRM instruction operating modes

MODE	Description
0	Shows the component that triggered the interrupt in the output parameter ID and sets the output parameter NEW to TRUE.
1	Describes all output parameters, independent of the interrupt triggering component.
2	Checks whether the component specified in input parameter F_ID has triggered the interrupt. • If not, NEW = FALSE • If yes, NEW = TRUE, and all other outputs parameters are described.

Note

If you assign a destination area for TINFO or AINFO that is too short, RALRM cannot return the full information. Refer to the online information system of STEP 7 for immediate access to information on how to interpret the TINFO and AINFO returned buffers.

8.3.3 STATUS parameter for RDREC, WRREC, and RALRM

The output parameter STATUS contains error information that is interpreted as ARRAY[1...4] OF BYTE, with the following structure:

Table 8- 76 STATUS output array

Array element	Name	Description
STATUS[1]	Function_Num	• B#16#00, if no error • Function ID from DPV1-PDU: If an error occurs, B#16#80 is OR'ed (for read data record: B#16#DE; for write data record: B#16#DF). If no DPV1 protocol element is used, then B#16#C0 will be output.
STATUS[2]	Error_Decode	Location of the error ID
STATUS[3]	Error_Code_1	Error ID
STATUS[4]	Error_Code_2	Manufacturer-specific error ID expansion

Table 8- 77 STATUS[2] values

Error_decode (B#16#....)	Source	Description
00 to 7F	CPU	No error or no warning
80	DPV1	Error according to IEC 61158-6
81 to 8F	CPU	B#16#8x shows an error in the "xth" call parameter of the instruction.
FE, FF	DP Profile	Profile-specific error

Table 8- 78 STATUS[3] values

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
00	00		No error, no warning
70	00	Reserved, reject	Initial call; no active data record transfer
	01	Reserved, reject	Initial call; data record transfer has started
	02	Reserved, reject	Intermediate call; data record transfer already active
80	90	Reserved, pass	Invalid logical start address
	92	Reserved, pass	Illegal type for Variant pointer

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
	93	Reserved, pass	The DP component addressed via ID or F_ID is not configured.
	96		The "RALRM (Page 263)" cannot supply the OB start information, management information, header information, or additional interrupt information. For OBs 4x, 55, 56, 57, 82, and 83, you can use the "DPNRM_DG (Page 271)" instruction to read the current diagnostics message frame of the relevant DP slave asynchronously (address information from OB start information).
	A0	Read error	Negative acknowledgement while reading from the module
	A1	Write error	Negative acknowledgement while writing to the module
	A2	Module failure	DP protocol error at layer 2 (for example, slave failure or bus problems)
	A3	Reserved, pass	• PROFIBUS DP: DP protocol error with Direct-Data-Link-Mapper or User-Interface/User • PROFINET IO: General CM error
	A4	Reserved, pass	Communication on the communication bus disrupted
	A5	Reserved, pass	-
	A7	Reserved, pass	DP slave or modules is occupied (temporary error).
	A8	Version conflict	DP slave or module reports non-compatible versions.
	A9	Feature not supported	Feature not supported by DP slave or module
	AA to AF	User specific	DP slave or module reports a manufacturer-specific error in its application. Please check the documentation from the manufacturer of the DP slave or module.
	B0	Invalid index	Data record not known in module; illegal data record number ≥ 256
	B1	Write length error	The length information in the RECORD parameter is incorrect. • With "RALRM": Length error in AINFO **Note**: Refer to the online information system of STEP 7 for immediate access to information on how to interpret the "AINFO" returned buffers. • With "RDREC (Page 260)" and "WRREC (Page 260)": Length error in "MLEN"
	B2	Invalid slot	The configured slot is not occupied.
	B3	Type conflict	Actual module type does not match specified module type.
	B4	Invalid area	DP slave or module reports access to an invalid area.
	B5	Status conflict	DP slave or module not ready
	B6	Access denied	DP slave or module denies access.
	B7	Invalid range	DP slave or module reports an invalid range for a parameter or value.
	B8	Invalid parameter	DP slave or module reports an invalid parameter.

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
	B9	Invalid type	DP slave or module reports an invalid type: • With "RDREC (Page 260)": Buffer too small (subsets cannot be read) • With "WRREC (Page 260)": Buffer too small (subsets cannot be written)
	BA to BF	User specific	DP slave or module reports a manufacturer-specific error when accessing. Please check the documentation from the manufacturer of the DP slave or module.
	C0	Read constraint conflict	• With "WRREC (Page 260)": The data can only be written when the CPU is in STOP mode. Note: This means that data cannot be written by the user program. You can only write the data online with a PG/PC. • With "RDREC (Page 260)": The module routes the data record, but either no data is present or the data can only be read when the CPU is in STOP mode. Note: If data can only be read when the CPU is in STOP mode, no evaluation by the user program is possible. In this case, you can only read the data online with a PG/PC.
	C1	Write constraint conflict	The data of the previous write request to the module for the same data record has not yet been processed by the module.
	C2	Resource busy	The module is currently processing the maximum possible number of jobs for a CPU.
	C3	Resource unavailable	The required operating resources are currently occupied.
	C4		Internal temporary error. Job could not be carried out. Repeat the job. If this error occurs often, check your installation for sources of electrical interference.
	C5		DP slave or module not available
	C6		Data record transfer was cancelled due to priority class cancellation.
	C7		Job aborted due to warm or cold restart on the DP master.
	C8 to CF		DP slave or module reports a manufacturer-specific resource error. Please check the documentation from the manufacturer of the DP slave or module.
	Dx	User specific	DP Slave specific. Refer to the description of the DP Slave.
81	00 to FF		Error in the initial call parameter (with "RALRM (Page 263)": MODE)
	00		Illegal operating mode
82	00 to FF		Error in the second call parameter

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
88	00 to FF		Error in the eighth call parameter (with "RALRM (Page 263)": TINFO) Note: Refer to the online information system of STEP 7 for immediate access to information on how to interpret the "TINFO" returned buffers.
	01		Wrong syntax ID
	23		Quantity structure exceeded or destination area too small
	24		Wrong range ID
	32		DB/DI number out of user range
	3A		DB/DI number is NULL for area ID DB/DI, or specified DB/DI does not exist.
89	00 to FF		Error in the ninth call parameter (with "RALRM (Page 263)": AINFO) Note: Refer to the online information system of STEP 7 for immediate access to information on how to interpret the "AINFO" returned buffers.
	01		Wrong syntax ID
	23		Quantity structure exceeded or destination area too small
	24		Wrong range ID
	32		DB/DI number out of user range
	3A		DB/DI number is NULL for area ID DB/DI, or specified DB/DI does not exist.
8A	00 to FF		Error in the 10th call parameter
8F	00 to FF		Error in the 15th call parameter
FE, FF	00 to FF		Profile-specific error

Array element STATUS[4]

With DPV1 errors, the DP Master passes on STATUS[4] to the CPU and to the instruction. Without a DPV1 error, this value is set to 0, with the following exceptions for the RDREC:

- STATUS[4] contains the target area length from RECORD, if MLEN > the destination area length from RECORD.
- STATUS[4]=MLEN, if the actual data record length < MLEN < the destination area length from RECORD.
- STATUS[4]=0, if STATUS[4] > 255; would have to be set

In PROFINET IO, STATUS[4] has the value 0.

8.3.4 DPRD_DAT and DPWR_DAT

You can use the DPRD_DAT (Read consistent data) and DPWR_DAT (Write consistent data) instructions with PROFINET and PROFIBUS.

Table 8- 79 DPRD_DAT and DPWR_DAT instructions

LAD / FBD	SCL	Description

	`ret_val := DPRD_DAT(`
	`laddr:=_word_in_,`
	`record=>_variant_out_);`
		Use the DPRD_DAT instruction to read the consistent data of a DP standard slave/PROFINET IO device. If no errors occur during the data transfer, the data read is entered into the target area set up by the RECORD parameter. The target area must have the same length as you configured with STEP 7 for the selected module. When you call the DPRD_DAT instruction, you can only access the data of one module / DP identification under the configured start address.

	`ret_val := DPWR_DAT(`
	`laddr:=_word_in_,`
	`record:=_variant_in_);`
		Use the DPWR_DAT instruction to transfer the data in RECORD consistently to the addressed DP standard slave/PROFINET IO device. The source area must have the same length as you configured with STEP 7 for the selected module.

The CPU supports up to 64 bytes of consistent data. For consistent data areas greater than 64 bytes, the DPRD_DAT and DPWR_DAT instructions must be used. If required, these instructions can be used for data areas of 1 byte or greater. If access is rejected, error code W#16#8090 will result.

Note

If you are using the DPRD_DAT and DPWR_DAT instructions with consistent data, you must remove this consistent data from the process-image automatic update. Refer to "PLC concepts: Execution of the user program" (Page 61) for more information.

Table 8- 80 Data types for the parameters

Parameter and type		Data type	Description
LADDR	IN	HW_IO (Word)	• Configured start address from the "I" area of the module from which the data will be read (DPRD_DAT) • Configured start address from the process image output area of the module to which the data will be written (DPWR_DAT) Addresses have to be entered in hexadecimal format (for example, an input or output address of 100 means: LADDR:=W#16#64).
RECORD	OUT	Variant	Destination area for the user data that were read (DPRD_DAT) or source area for the user data to be written (DPWR_DAT). This must be exactly as large as you configured for the selected module with STEP 7. Only the data type Byte is permitted.
RET_VAL	OUT	Int	If an error occurs while the function is active, the return value contains an error code.

DPRD_DAT operations

The destination area must have the same length as configured for the selected module with STEP 7. If no error occurs during the data transfer, the data that have been read are entered into the destination area identified by RECORD.

If you read from a DP standard slave with a modular design or with several DP identifiers, you can only access the data of one module/DP identifier for each DPRD_DAT instruction call, specifying the configured start address.

DPWR_DAT operations

You transfer the data in RECORD consistently to the addressed DP standard slave/PROFINET IO. The data is transferred synchronously, that is, the write process is completed when the instruction is completed.

The source area must have the same length as you configured for the selected module with STEP 7.

If the DP standard slave has a modular design, you can only access one module of the DP slave.

Table 8- 81 DPRD_DAT and DPWR_DAT error codes

Error code	Description
0000	No error occurred
808x	System error with external DP interface module
8090	One of the following cases apply: • You have not configured a module for the specified logical base address. • You have ignored the restriction concerning the length of consistent data. • You have not entered the start address in the LADDR parameter in hexadecimal format.
8092	A type other than Byte is specified in the Any reference.
8093	No DP module/PROFINET IO device from which you can read (DPRD_DAT) or to which you can write (DPWR_DAT) consistent data exists at the logical address specified in LADDR.
80A0	Access error detected while the I/O devices were being accessed (DPRD_DAT).
80A1	Access error detected while the I/O devices were being accessed (DPWR_DAT).
80B0	Slave failure on external DP interface module
80B1	The length of the specified destination (DPRD_DAT) or source (DPWR_DAT) area is not identical to the user data length configured with STEP 7 Basic.
80B2, 80B3, 80C2, 80Fx	System error with external DP interface module (DPRD_DAT) and (DPWR_DAT)
87xy, 808x	System error with external DP interface module (DPRD_DAT)
85xy	System error with external DP interface module (DPWR_DAT)
80C0	The data have not yet been read by the module (DPRD_DAT).
80C1	The data of the previous write job on the module have not yet been processed by the module (DPWR_DAT).
8xyy[1]	General error information

Refer to "Extended instructions, Distributed I/O: Error information for RDREC, WRREC, and RALRM" (Page 265) for more information on general error codes.

Note

If you access DPV1 slaves, error information from these slaves can be forwarded from the DP master to the instruction.

8.3.5 DPNRM_DG

You can use the DPNRM_DG (Read diagnostic data) instruction with PROFIBUS.

Table 8- 82 DPNRM_DG instruction

LAD / FBD	SCL	Description

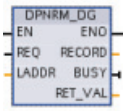
	`ret_val := DPNRM_DG(`
	`req:=_bool_in_,`
	`laddr:=_word_in_,`
	`record=>_variant_out_,`
	`busy=>_bool_out_);`
		Use the DPNRM_DG instruction to read the current diagnostic data of a DP slave in the format specified by EN 50 170 Volume 2, PROFIBUS. The data that has been read is entered in the destination area indicated by RECORD following error-free data transfer.

Table 8- 83 DPNRM_DG instruction data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	REQ=1: Read request
LADDR	IN	HW_DPSLAVE	Configured diagnostic address of the DP slave: Must be the address of the station and not for the I/O device. Select the station (and not the image of the device) in the "Network" view of the "Device configuration" to determine the diagnostic address. Enter the addresses in hexadecimal format. For example, diagnostic address 1022 means LADDR:=W#16#3FE.
RET_VAL	OUT	Int	If an error occurs while the function is active, the return value contains an error code. If no error occurs, the length of the data actually transferred is entered in RET_VAL.
RECORD	OUT	Variant	Destination area for the diagnostic data that were read. Only the Byte data type is permitted. The minimum length of the data record to be read or the destination area is 6. The maximum length of the data record to be sent is 240. Standard slaves can provide more than 240 bytes of diagnostic data up to a maximum of 244 bytes. In this case, the first 240 bytes are transferred to the destination area, and the overflow bit is set in the data.
BUSY	OUT	Bool	BUSY=1: The read job is not yet completed

You start the read job by assigning 1 to the input parameter REQ in the DPNRM_DG instruction call. The read job is executed asynchronously, in other words, it requires several DPNRM_DG instruction calls. The status of the job is indicated by the output parameters RET_VAL and BUSY.

Table 8- 84 Slave diagnostic data structure

Byte	Description
0	Station status 1
1	Station status 2
2	Station status 3
3	Master station number
4	Vendor ID (high byte)
5	Vendor ID (low byte)
6 ...	Additional slave-specific diagnostic information

Table 8- 85 DPNRM_DG instruction error codes

Error code	Description	Restriction
0000	No error	-
7000	First call with REQ=0: No data transfer active; BUSY has the value 0.	-
7001	First call with REQ =1: No data transfer active; BUSY has the value 1.	Distributed I/Os
7002	Interim call (REQ irrelevant): Data transfer already active; BUSY has the value 1.	Distributed I/Os
8090	Specified logical base address invalid: There is no base address.	-
8092	The type specified in the Any reference is not Byte.	-
8093	• This instruction is not permitted for the module specified by LADDR (S7-DP modules for S7-1200 are permitted). • LADDR specifies the I/O device instead of specifying the station. Select the station (and not the image of the device) in the "Network" view of the "Device configuration" to determine the diagnostic address for LADDR.	-
80A2	• DP protocol error at layer 2 (for example, slave failure or bus problems) • For ET200S, data record cannot be read in DPV0 mode.	Distributed I/Os
80A3	DP protocol error with user interface/user	Distributed I/Os
80A4	Communication problem on the communication bus	The error occurs between the CPU and the external DP interface module.
80B0	• The instruction is not possible for module type. • The module does not recognize the data record. • Data record number 241 is not permitted.	-
80B1	The length specified in the RECORD parameter is incorrect.	Specified length > record length
80B2	The configured slot is not occupied.	-
80B3	Actual module type does not match the required module type.	-
80C0	There is no diagnostic information.	-
80C1	The data of the previous write job for the same data record on the module have not yet been processed by the module.	-

Error code	Description	Restriction
80C2	The module is currently processing the maximum possible number of jobs for a CPU.	-
80C3	The required resources (memory, etc.) are currently occupied.	-
80C4	Internal temporary error. The job could not be processed. Repeat the job. If this error occurs frequently, check your system for electrical disturbance sources.	-
80C5	Distributed I/Os not available	Distributed I/Os
80C6	Data record transfer was stopped due to a priority class abort (restart or background)	Distributed I/Os
8xyy[1]	General error codes	

Refer to "Extended instructions, Distributed I/O: Error information for RDREC, WRREC, and RALRM" (Page 265) for more information on general error codes.

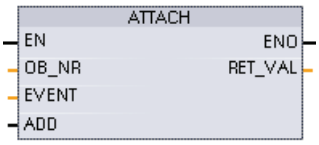
8.4 Interrupts

8.4.1 Attach and detach instructions

You can activate and deactivate interrupt event-driven subprograms with the ATTACH and DETACH instructions.

Table 8- 86 ATTACH and DETACH instructions

LAD / FBD	SCL	Description

	`ret_val := ATTACH(`
	`ob_nr:=_int_in_,`
	`event:=_event_att_in_,`
	`add:=_bool_in_);`
		ATTACH enables interrupt OB subprogram execution for a hardware interrupt event.

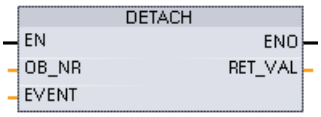
	`ret_val := DETACH(`
	`ob_nr:=_int_in_,`
	`event:=_event_att_ in);`
		DETACH disables interrupt OB subprogram execution for a hardware interrupt event.

Table 8- 87 Data types for the parameters

Parameter and type		Data type	Description
OB_NR	IN	OB_ATT	Organization block identifier: Select from the available hardware interrupt OBs that were created using the "Add new block" feature. Double-click on the parameter field, then click on the helper icon to see the available OBs.
EVENT	IN	EVENT_ATT	Event identifier: Select from the available hardware interrupt events that were enabled in PLC device configuration for digital inputs or high-speed counters. Double-click on the parameter field, then click on the helper icon to see the available events.
ADD (ATTACH only)	IN	Bool	• ADD = 0 (default): This event replaces all previous event attachments for this OB. • ADD = 1: This event is added to previous event attachments for this OB.
RET_VAL	OUT	Int	Execution condition code

Hardware interrupt events

The following hardware interrupt events are supported by the CPU:

- Rising edge events (all built-in CPU digital inputs and SB digital inputs)
  - A rising edge occurs when the digital input transitions from OFF to ON as a response to a change in the signal from a field device connected to the input.
- Falling edge events (all built-in CPU digital inputs and SB digital inputs)
  - A falling edge occurs when the digital input transitions from ON to OFF.
- High-speed counter (HSC) current value = reference value (CV = RV) events (HSC 1 through 6)
  - A CV = RV interrupt for a HSC is generated when the current count transitions from an adjacent value to the value that exactly matches a reference value that was previously established.
- HSC direction changed events (HSC 1 through 6)
  - A direction changed event occurs when the HSC is detected to change from increasing to decreasing, or from decreasing to increasing.
- HSC external reset events (HSC 1 through 6)
  - Certain HSC modes allow the assignment of a digital input as an external reset that is used to reset the HSC count value to zero. An external reset event occurs for such a HSC, when this input transitions from OFF to ON.

Enabling hardware interrupt events in the device configuration

Hardware interrupts must be enabled during the device configuration. You must check the enable-event box in the device configuration for a digital input channel or a HSC, if you want to attach this event during configuration or run time.

Check box options within the PLC device configuration:

- Digital input
  - Enable rising edge detection
  - Enable falling edge detection
- High-speed counter (HSC)
  - Enable this high-speed counter for use
  - Generate interrupt for counter value equals reference value count
  - Generate interrupt for external reset event
  - Generate interrupt for direction change event

Adding new hardware interrupt OB code blocks to your program

By default, no OB is attached to an event when the event is first enabled. This is indicated by the "HW interrupt:" device configuration "<not connected>" label. Only hardware-interrupt OBs can be attached to a hardware interrupt event. All existing hardware-interrupt OBs appear in the "HW interrupt:" drop-down list. If no OB is listed, then you must create an OB of type "Hardware interrupt" as follows. Under the project tree "Program blocks" branch:

1. Double-click "Add new block", select "Organization block (OB)" and choose "Hardware interrupt".
2. Optionally, you can rename the OB, select the programming language (LAD or FBD), and select the block number (switch to manual and choose a different block number than that suggested).
3. Edit the OB and add the programmed reaction that you want to execute when the event occurs. You can call FCs and FBs from this OB, to a nesting depth of four.

OB_NR parameter

All existing hardware-interrupt OB names appear in the device configuration "HW interrupt:" drop-down list and in the ATTACH / DETACH parameter OB_NR drop-list.

EVENT parameter

When a hardware interrupt event is enabled, a unique default event name is assigned to this particular event. You can change this event name by editing the "Event name:" edit box, but it must be a unique name. These event names become tag names in the "Constants" tag table, and appear on the EVENT parameter drop-down list for the ATTACH and DETACH instruction boxes. The value of the tag is an internal number used to identify the event.

General operation

Each hardware event can be attached to a hardware-interrupt OB which will be queued for execution when the hardware interrupt event occurs. The OB-event attachment can occur at configuration time or at run time.

You have the option to attach or detach an OB to an enabled event at configuration time. To attach an OB to an event at configuration time, you must use the "HW interrupt:" drop-down list (click on the down arrow on the right) and select an OB from the list of available hardware-interrupt OBs. Select the appropriate OB name from this list, or select "<not connected>" to remove the attachment.

You can also attach or detach an enabled hardware interrupt event during run time. Use the ATTACH or DETACH program instructions during run time (multiple times if you wish) to attach or detach an enabled interrupt event to the appropriate OB. If no OB is currently attached (either from a "<not connected>" selection in device configuration, or as a result of executing a DETACH instruction), the enabled hardware interrupt event is ignored.

DETACH operation

Use the DETACH instruction to detach either a particular event or all events from a particular OB. If an EVENT is specified, then only this one event is detached from the specified OB_NR; any other events currently attached to this OB_NR will remain attached. If no EVENT is specified, then all events currently attached to OB_NR will be detached.

Condition codes

Table 8- 88 Condition codes

RET_VAL (W#16#....)	ENO	Description
0000	1	No error
0001	1	Nothing to Detach (DETACH only)
8090	0	OB does not exist
8091	0	OB is wrong type
8093	0	Event does not exist

8.4.2 Cyclic interrupts

8.4.2.1 SET_CINT (Set cyclic interrupt)

Table 8- 89 SET_CINT (Set cyclic interrupt instruction)

LAD / FBD	SCL	Description

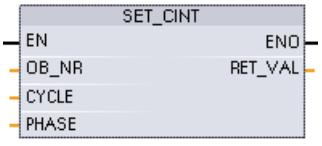
	`ret_val := SET_CINT(`
	`  ob_nr:=_int_in_,`
	`  cycle:=_udint_in_,`
	`  phase:=_udint_in_);`
		Set the specified interrupt OB to begin cyclic execution that interrupts the program scan.

Table 8- 90 Data types for the parameters

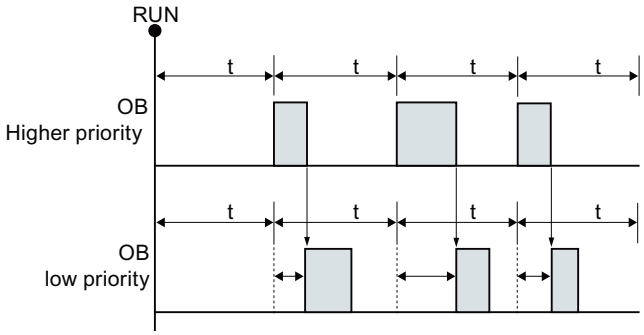
Parameter and type		Data type	Description
OB_NR	IN	OB_CYCLIC	OB number (accepts symbolic name)
CYCLE	IN	UDInt	Time interval, in microseconds
PHASE	IN	UDInt	Phase shift, in microseconds
RET_VAL	OUT	Int	Execution condition code

Time parameter examples:

- If the CYCLE time = 100 us, then the interrupt OB referenced by OB_NR interrupts the cyclic program scan every 100 us. The interrupt OB executes and then returns execution control to the program scan, at the point of interruption.
- If the CYCLE time = 0, then the interrupt event is deactivated and the interrupt OB is not executed.
- The PHASE (phase shift) time is a specified delay time that occurs before the CYCLE time interval begins. You can use the phase shift to control the execution timing of lower priority OBs.

If lower and higher priority OBs are called in the same time interval, the lower priority OB is only called after the higher priority OB has finished processing. The execution start time for the low priority OB can shift depending on the processing time of higher priority OBs.

OB call without phase shift



If you want to start the execution of a lower priority OB on a fixed time cycle, then phase shift time should be greater then the processing time of higher priority OBs.

OB-call with phase shift

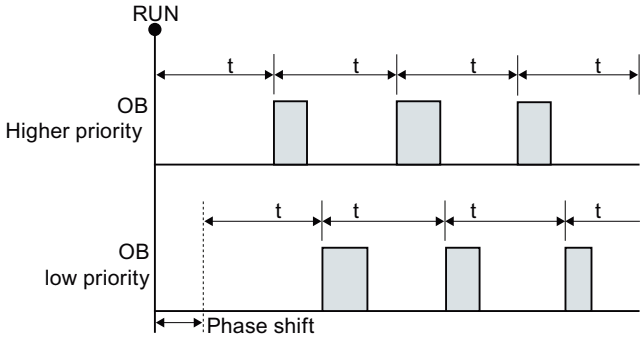


Table 8- 91 Condition codes

RET_VAL (W#16#....)	Description
0000	No error
8090	OB does not exist or is of wrong type
8091	Invalid cycle time
8092	Invalid phase shift time
80B2	OB has no attached event

8.4.2.2 QRY_CINT (Query cyclic interrupt)

Table 8- 92 QRY_CINT (Query cyclic interrupt)

LAD / FBD	SCL	Description

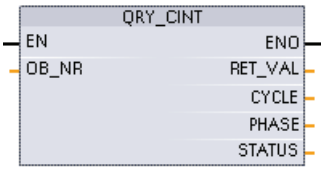
	`ret_val := QRY_CINT(`
	`ob_nr:=_int_in_,`
	`cycle=>_udint_out_,`
	`phase=>_udint_out__,`
	`status=>_word_out_);`
		Get parameter and execution status from a cyclic interrupt OB. The values that are returned existed at the time QRY_CINT was executed.

Table 8- 93 Data types for the parameters

Parameter and type		Data type	Description
OB_NR	IN	OB_CYCLIC	OB number (accepts symbolic name like OB_MyOBName)
RET_VAL	OUT	Int	Execution condition code
CYCLE	OUT	UDInt	Time interval, in microseconds
PHASE	OUT	UDInt	Phase shift, in microseconds
STATUS	OUT	Word	Cyclic interrupt status code: • Bits 0 to 4, see the STATUS table below • Other bits, always 0

Table 8- 94 STATUS parameter

Bit	Value	Description
0	0	During CPU RUN
	1	During startup
1	0	The interrupt is enabled.
	1	Interrupt is disabled via the DIS_IRT instruction.
2	0	The interrupt is not active or has elapsed.
	1	The interrupt is active.
4	0	The OB identified by OB_NR does not exist.
	1	The OB identified by OB_NR exists.

Bit	Value	Description
Other Bits		Always 0

If an error occurs, RET_VAL displays the appropriate error code and the parameter STATUS = 0.

Table 8- 95 RET_VAL parameter

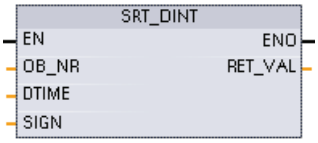
RET_VAL (W#16#....)	Description
0000	No error
8090	OB does not exist or is of wrong type.
80B2	OB has no attached event.

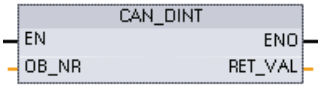
8.4.3 Time delay interrupts

You can start and cancel time delay interrupt processing with the SRT_DINT and CAN_DINT instructions, or query the interrupt status with the QRY_DINT instruction. Each time delay interrupt is a one-time event that occurs after the specified delay time. If the time delay event is cancelled before the time delay expires, the program interrupt does not occur.

Table 8- 96 SRT_DINT, CAN_DINT, and QRY_DINT instructions

LAD / FBD	SCL	Description

	`ret_val := SRT_DINT(`
	`ob_nr:=_int_in_,`
	`dtime:=_time_in_,`
	`sign:=_word_in_);`
		SRT_DINT starts a time delay interrupt that executes an OB when the delay time specified by parameter DTIME has elapsed.

	`ret_val := CAN_DINT(` `ob_nr:=_int_in_);`	CAN_DINT cancels a time delay interrupt that has already started. The time delay interrupt OB is not executed in this case.

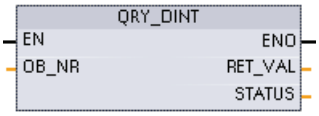
	`ret_val := QRY_DINT(`
	`ob_nr:=_int_in_,`
	`status=>_word_out_);`
		QRY_DINT queries the status of the time delay interrupt specified by the OB_NR parameter.

Table 8- 97 Data types for the parameters

Parameter and type		Data type	Description
OB_NR	IN	OB_DELAY	Organization block (OB) to be started after a time-delay: Select from the available time-delay interrupt OBs that were created using the "Add new block" project tree feature. Double-click on the parameter field, then click on the helper icon to see the available OBs.
DTIME [1]	IN	Time	Time delay value (1 to 60000 ms)

Parameter and type		Data type	Description
SIGN [1]	IN	Word	Not used by the S7-1200: Any value is accepted. A value must be assigned to prevent errors.
RET_VAL	OUT	Int	Execution condition code
STATUS	OUT	Word	QRY_DINT instruction: Status of the specified time-delay interrupt OB, see the table below

[1] Only for SRT_DINT

Operation

The SRT_DINT instruction specifies a time delay, starts the internal time delay timer, and associates a time delay interrupt OB subprogram with the time delay timeout event. When the specified time delay has elapsed, a program interrupt is generated that triggers the execution of the associated time delay interrupt OB. You can cancel an in-process time delay interrupt before the specified time delay occurs by executing the CAN_DINT instruction. The total number of active time delay and cyclic interrupt events must not exceed four.

Adding time delay interrupt OB subprograms to your project

Only time delay interrupt OBs can be assigned to the SRT_DINT and CAN_DINT instructions. No time delay interrupt OB exists in a new project. You must add time delay interrupt OBs to your project. To create a time-delay interrupt OB, follow these steps:

1. Double-click the "Add new block" item in the "Program blocks" branch of the project tree, select "Organization block (OB)", and choose "Time delay interrupt".
2. You have the option to rename the OB, select the programming language, or select the block number. Switch to manual numbering if you want to assign a different block number than the number that was assigned automatically.
3. Edit the time delay interrupt OB subprogram and create programmed reaction that you want to execute when the time delay timeout event occurs. You can call other FC and FB code blocks from the time delay interrupt OB, with a maximum nesting depth of four.
4. The newly assigned time delay interrupt OB names will be available when you edit the OB_NR parameter of the SRT_DINT and CAN_DINT instructions.

QRY_DINT parameter STATUS

Table 8- 98 If there is an error (REL_VAL <> 0), then STATUS = 0.

Bit	Value	Description
0	0	In RUN
	1	In startup
1	0	The interrupt is enabled.
	1	The interrupt is disabled.
2	0	The interrupt is not active or has elapsed.
	1	The interrupt is active.

Bit	Value	Description
4	0	An OB with an OB number given in OB_NR does not exist.
	1	An OB with an OB number given in OB_NR exists.
Other bits		Always 0

Condition codes

Table 8- 99 Condition codes for SRT_DINT, CAN_DINT, and QRY_DINT

RET_VAL (W#16#...)	Description
0000	No error occurred
8090	Incorrect parameter OB_NR
8091	Incorrect parameter DTIME
80A0	Time delay interrupt has not started.

8.4.4 Asynchronous event interrupts

Use the DIS_AIRT and EN_AIRT instructions to disable and enable alarm interrupt processing.

Table 8- 100 DIS_AIRT and EN_AIRT instructions

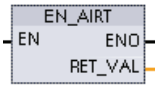
LAD / FBD	SCL	Description
	`DIS_AIRT();`	DIS_AIRT delays the processing of new interrupt events. You can execute DIS_AIRT more than once in an OB.
	`EN_AIRT();`	EN_AIRT enables the processing of interrupt events that you previously disabled with the DIS_AIRT instruction. Each DIS_AIRT execution must be cancelled by an EN_AIRT execution. The EN_AIRT executions must occur within the same OB, or any FC or FB called from the same OB, before interrupts are enabled again for this OB.

Table 8- 101 Data types for the parameters

Parameter and type		Data type	Description
RET_VAL	OUT	Int	Number of delays = number of DIS_AIRT executions in the queue.

The DIS_AIRT executions are counted by the operating system. Each of these remains in effect until it is cancelled again specifically by an EN_AIRT instruction, or until the current OB has been completely processed. For example: if you disabled interrupts five times with five DIS_AIRT executions, you must cancel these with five EN_AIRT executions before interrupts become enabled again.

After the interrupt events are enabled again, the interrupts that occurred while DIS_AIRT was in effect are processed, or the interrupts are processed as soon as the current OB has been executed.

Parameter RET_VAL indicates the number of times that interrupt processing was disabled, which is the number of queued DIS_AIRT executions. Interrupt processing is only enabled again when parameter RET_VAL = 0.

8.5 Diagnostics (PROFINET or PROFIBUS)

8.5.1 LED instruction

Table 8- 102 LED instruction

LAD / FBD	SCL	Description

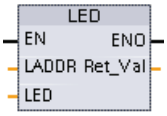
	`ret_val := LED(`
	`  laddr:=_word_in_,`
	`  LED:=_uint_in_);`
		Use the LED instruction to read the state of the LEDs on a CPU or interface. The specified LED state is returned by the RET_VAL output.

Table 8- 103 Data types for the parameters

Parameter and type		Data type	Description		
LADDR	IN	HW_IO	Identification number of the CPU or interface[1]		
LED	IN	UInt	LED identifier number		
			1	RUN/STOP	Color 1 = green, color 2 = yellow
			2	Error	Color 1 = red
			3	Maintenance	Color 1 = yellow
			4	Redundancy	Not applicable
			5	Link	Color 1 = green
			6	Tx/Rx	Color 1 = yellow
RET_VAL	OUT	Int	Status of the LED		

[1] For example, you can select the CPU (such as "PLC_1") or the PROFINET interface from the drop-down list of the parameter.

Table 8- 104 Status of RET_VAL

RET_VAL (W#16#...)	Description	
0 to 9 LED state	0	LED does not exist
	1	Off
	2	Color 1 On (solid)

RET_VAL (W#16#...)	Description	
	3	Color 2 On (Solid)
	4	Color 1 flashing at 2 Hz
	5	Color 2 flashing 2 Hz
	6	Color 1 & 2 flashing alternatively at 2 Hz
	7	Color 1 on (Tx/Rx)
	8	Color 2 on (Tx/Rx)
	9	State of the LED is not available
8091	Device identified by LADDR does not exist	
8092	Device identified by LADDR does not support LEDs	
8093	LED identifier not defined	
80Bx	CPU identified by LADDR does not support the LED instruction	

8.5.2 DeviceStates instruction

Table 8- 105 DeviceStates instruction

LAD / FBD	SCL	Description

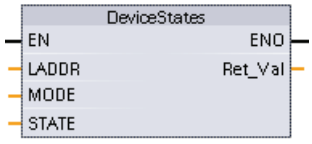
	`ret_val := DeviceStates(`
	`  laddr:=hw_io_in_,`
	`  mode:=_uint_in_,`
	`  state:=_variant_inout_);`
		Retrieves the I/O device operational states of an I/O subsystem. This information corresponds with the STEP 7 diagnostics view.

Table 8- 106 Data types for the parameters

Parameter and type		Data type	Description
LADDR	IN	HW_IOSYSTEM	Logical address: (Identifier for the I/O system)
MODE	IN	UInt	Status type: • 1: Configured stations • 2: Defective stations • 3: Deactivated stations • 4: Existing stations
RET_VAL	OUT	Int	Execution condition code
STATE[1]	InOut	Variant	Buffer containing the error status of each device: • Summary bit: Bit 0 =1, if one of the state bits of the I/O devices is 1 • State bit: State of I/O device with station number *n* according to the selected MODE. For example, MODE = 2 and bit 3 = 1 means station 3 is faulty.

[1] For PROFIBUS-DP, the length of the status information is 128 bits. For PROFIBUS I/O, the length is 1024 bits.

After execution, the STATE parameter contains the error state of each I/O device as a bit list (for the specified LADDR and MODE).

The data type used for the STATE parameter can be any bit type (Bool, Byte, Word, or DWord) or an array of a bit type.

Table 8- 107 Condition codes

RET_VAL (W#16#...)	Description
0	No error
8091	LADDR does not exist.
8092	LADDR does not address an I/O system.
80Bx	DeviceStates instruction not supported by the CPU for this LADDR.
8452	The complete state data is too large for STATE. The STATE parameter contains a partial result.

8.5.3 ModuleStates instruction

Table 8- 108 ModuleStates instruction

LAD / FBD	SCL	Description

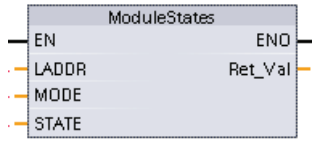
	`ret_val := ModuleStates(`
	`laddr:=_word_in_,`
	`mode:=_uint_in,`
	`state:=_variant_inout);`
		Retrieves the I/O module operational states of I/O devices. This information corresponds with the STEP 7 diagnostics view.

Table 8- 109 Data types for the parameters

Parameter and type		Data type	Description
LADDR	IN	HW_IOSYSTEM	Logical address (Identifier for the I/O device)
MODE	IN	UInt	Status type: • 1: Configured modules • 2: Defective modules • 3: Deactivated modules • 4: Existing modules

Parameter and type		Data type	Description
RET_VAL	OUT	Int	Status (condition code)
STATE[1]	InOut	Variant	Buffer containing the error status of each device • Summary bit: Bit 0 =1, if one of the state bits of the I/O devices is 1 • State bit: State of I/O device with station number *n* according to the selected MODE. For example, MODE = 2 and bit 3 = 1 means station 3 is faulty.

[1] The length required is dependent on the I/O device, with a maximum of 128 bits.

Table 8- 110 Condition codes

RET_VAL (W#16#...)	Description
0	No error
8091	Device identified by LADDR does not exist.
8092	Device identified by LADDR does not address an I/O device.
80Bx	ModuleStates instruction not supported by this CPU for this LADDR.
8452	The complete state data is too large for STATE. The STATE parameter contains a partial result.

8.5.4 GET_DIAG instruction

Table 8- 111 GET_DIAG instruction

LAD / FBD	SCL	Description

	`ret_val := GET_DIAG(`
	`  mode:=_uint_in_,`
	`  laddr:=_word_in_,`
	`  cnt_diag=>_uint_out_,`
	`  diag:=_variant_inout_,`
	`  detail:=_variant_inout_);`
		Reads the diagnostic information from a specified hardware device.

Table 8- 112 Data types for the parameters

Parameter and type		Data type	Description
MODE	IN	UInt	Mode
LADDR	IN	HW_ANY (Word)	Identification number of the hardware device
DIAG	InOut	Variant	Diagnostic information according to diagnostic mode
DETAIL	InOut	Variant	Diagnostic detail according to diagnostic mode
RET_VAL	OUT	Int	Execution result / error message
CNT_DIAG	OUT	UInt	Count of the returned diagnostic details

Input LADDR selects the hardware device. The type of the delivered diagnostic information is selected by the input MODE.

Table 8- 113 MODE parameter

MODE input	DIAG output	CNT_DIAG output	DETAIL output
0	Bit list of supported modes as DWord	0	Nothing
1	Diagnostic state as diagnostic information source (DIS)	0	Nothing
2	Diagnostic navigation node (DNN)	0	Nothing

Table 8- 114 Structure of the diagnostic information source (DIS)

DIS: Struct;		
	OwnState:	UInt;
	MaintenanceState:	DWord;
	IOState:	Word;
	ComponentStateDetail:	DWord;
	OperatingState:	UInt;
	End_Struct	

Table 8- 115 Structure of the diagnostic navigation node (DNN)

DNN: Struct;		
	SubordinateState:	UInt;
	SubordinateIOState:	Word;
	DNNmode:	Word;
	End_Struct	

Table 8- 116 Condition codes

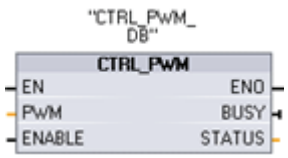
RET_VAL (W#16#...)	Description
0	No error
n	All of the n existing diagnostic details could not be provided.
8080	Mode not supported.
8081	Data type at parameter DIAG not supported with given mode.
8082	Data type at parameter DETAIL not supported with given mode.
8090	Device identified by LADDR does not exist.
8091	Channel does not exist.
80C1	Lack of resources for parallel executions.

8.6 Pulse

8.6.1 CTRL_PWM instruction

Table 8- 117 CTRL_PWM (Pulse Width Modulation) instruction

LAD / FBD	SCL	Description

	`"CTRL_PWM_DB"(`
	`PWM:=_word_in_,`
	`enable:=_bool_in_,`
	`busy=>_bool_out_,`
	`status=>_word_out_);`
		Provides a fixed cycle time output with a variable duty cycle. The PWM output runs continuously after being started at the specified frequency (cycle time). The pulse width is varied as required to affect the desired control.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "CTRL_PWM_DB" is the name of the instance DB.

CTRL_HSC

Table 8- 118 Data types for the parameters

Parameter and type		Data type	Description
PWM	IN	HW_PWM (Word)	PWM identifier: Names of enabled pulse generators will become tags in the "constant" tag table, and will be available for use as the PWM parameter. (Default value: 0)
ENABLE	IN	Bool	1=start pulse generator 0 = stop pulse generator
BUSY	OUT	Bool	Function busy (Default value: 0)
STATUS	OUT	Word	Execution condition code (Default value: 0)

The CTRL_PWM instruction stores the parameter information in the DB. The data block parameters are not separately changed by the user, but are controlled by the CTRL_PWM instruction.

Specify the enabled pulse generator to use, by using its tag name for the PWM parameter.

When the EN input is TRUE, the PWM_CTRL instruction starts or stops the identified PWM based on the value at the ENABLE input. Pulse width is specified by the value in the associated Q word output address.

Because the CPU processes the request when the CTRL_PWM instruction is executed, parameter BUSY will always report FALSE. If an error is detected, then ENO is set to FALSE, and parameter STATUS contains a condition code.

The pulse width will be set to the initial value configured in device configuration when the CPU first enters RUN mode. You write values to the Q-word location specified in device configuration ("Output addresses" / "Start address:") as needed to change the pulse width. You use an instruction such as a move, convert, math, or PID box to write the desired pulse width to the appropriate Q word. You must use the valid range for the Q-word value (percent, thousandths, ten-thousandths, or S7 analog format).

Note

Digital I/O points assigned to PWM and PTO cannot be forced

The digital I/O points used by the pulse-width modulation (PWM) and pulse-train output (PTO) devices are assigned during device configuration. When digital I/O point addresses are assigned to these devices, the values of the assigned I/O point addresses cannot be modified by the Watch table force function.

Table 8- 119 Value of the STATUS parameter

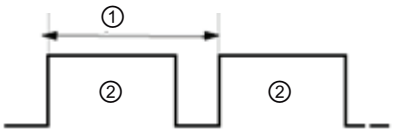
STATUS	Description
0	No error
80A1	PWM identifier does not address a valid PWM.

Table 8- 120 Common condition codes

Condition code[1]	Description
8022	Area too small for input
8023	Area too small for output
8024	Illegal input area
8025	Illegal output area
8028	Illegal input bit assignment
8029	Illegal output bit assignment
8030	Output area is a read-only DB.
803A	DB does not exist.

[1] If one of these errors occurs when a code block is executed, the CPU goes to STOP mode unless you use the GetError or GetErrorID instructions within that code block to create a programmed reaction to the error.

8.6.2 Operation of the pulse outputs



Pulse width can be expressed as hundredths of the cycle time (0 to 100), as thousandths (0 to 1000), as ten thousandths (0 to 10000), or as S7 analog format.

The pulse width can vary from 0 (no pulse, always off) to full scale (no pulse, always on).

① Cycle time

② Pulse width

Since the PWM output can be varied from 0 to full scale, it provides a digital output that in many ways is the same as an analog output. For example, the PWM output can be used to control the speed of a motor from stop to full speed, or it can be used to control position of a valve from closed to fully opened.

Two pulse generators are available for controlling high-speed pulse output functions: PWM and Pulse train output (PTO). PTO is used by the motion control instructions. You can assign each pulse generator to either PWM or PTO, but not both at the same time.

The two pulse generators are mapped to specific digital outputs as shown in the following table. You can use onboard CPU outputs, or you can use the optional signal board outputs. The output point numbers are shown in the following table (assuming the default output configuration). If you have changed the output point numbering, then the output point numbers will be those you assigned. Regardless, PTO1/PWM1 uses the first two digital outputs, and PTO2/PWM2 uses the next two digital outputs, either on the CPU or on the attached signal board. Note that PWM requires only one output, while PTO can optionally use two outputs per channel. If an output is not required for a pulse function, it is available for other uses.

NOTICE
Pulse-train outputs cannot be used by other instructions in the user program When you configure the outputs of the CPU or signal board as pulse generators (for use with the PWM or basic motion control instructions), the corresponding outputs addresses (Q0.0, Q0.1, Q4.0, and Q4.1) are removed from the Q memory and cannot be used for other purposes in your user program. If your user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.

Table 8- 121 Default output assignments for the pulse generators

Description		Pulse	Direction
PTO 1	Onboard CPU	Q0.0	Q0.1
	Signal board	Q4.0	Q4.1
PWM 1	Onboard CPU	Q0.0	--
	Signal board	Q4.0	--
PTO 2	Onboard CPU	Q0.2	Q0.3
	Signal board	Q4.2	Q4.3
PWM 2	Onboard CPU	Q0.2	--
	Signal board	Q4.2	--

8.6.3 Configuring a pulse channel for PWM

To prepare for PWM operation, first configure a pulse channel in the device configuration by selecting the CPU, then Pulse Generator (PTO/PWM), and choose either PWM1 or PWM2. Enable the pulse generator (check box). If a pulse generator is enabled, a unique default name is assigned to this particular pulse generator. You can change this name by editing it in the "Name:" edit box, but it must be a unique name. Names of enabled pulse generators will become tags in the "constant" tag table, and will be available for use as the PWM parameter of the CTRL_PWM instruction.

NOTICE
The maximum pulse frequency of the pulse output generators for the digital output is 100 KHz (for the CPU), 20 KHz (for a SB), or 200 KHz (for a high-speed SB). However, STEP 7 does not alert you when you configure an axis that with a maximum speed or frequency that exceeds this hardware limitation. This could cause problems with your application, so always ensure that you do not exceed the maximum pulse frequency of the hardware.

You have the option to rename the pulse generator, add a comment, and assign parameters as follows:

- Pulse generator used as follows: PWM or PTO (choose PWM)
- Output source: onboard CPU or SB
- Time base: milliseconds or microseconds
- Pulse width format:
  - Hundredths (0 to 100)
  - Thousandths (0 to 1000)
  - Ten-thousandths (0 to 10000)
  - S7 analog format (0 to 27648)

- Cycle time: Enter your cycle time value. This value can only be changed in Device configuration.
- Initial pulse width: Enter your initial pulse width value. The pulse width value can be changed during runtime.

Enter the start address to configure the output addresses. Enter the Q word address where you want to locate the pulse width value.

NOTICE
Pulse-train outputs cannot be used by other instructions in the user program
When you configure the outputs of the CPU or signal board as pulse generators (for use with the PWM or basic motion control instructions), the corresponding outputs addresses (Q0.0, Q0.1, Q4.0, and Q4.1) are removed from the Q memory and cannot be used for other purposes in your user program. If your user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.

The default location is QW1000 for PWM1, and QW1002 for PWM2. The value at this location controls the width of the pulse and is initialized to the "Initial pulse width:" value specified above each time the CPU transitions from STOP to RUN mode. You change this Q-word value during run time to cause a change in the pulse width.

8.7 Data logging

Your control program can use the Data log instructions to store run-time data values in persistent log files. The data log files are stored in flash memory (CPU or memory card). Log file data is stored in standard CSV (Comma Separated Value) format. The data records are organized as a circular log file of a pre-determined size.

The Data log instructions are used in your program to create, open, write a record, and close the log files. You decide which program values will be logged by creating a data buffer that defines a single log record. Your data buffer is used as temporary storage for a new log record. New current values must be programmatically moved into the buffer during run-time. When all of the current data values are updated, you can execute the DataLogWrite instruction to transfer data from the buffer to a data log record.

Use the built-in PLC Web server to manage your data log files. Download recent records, all records, clear records, or delete log files with the "Data Logs" standard web page. After a data log file is transferred to your PC, then you can analyze the data with standard spreadsheet tools like Microsoft Excel.

8.7.1 Data log record structure

The DATA and HEADER parameters of the DataLogCreate instruction assign the data type and the column header description of all data elements of a log record.

DATA parameter for the DataLogCreate instruction

The DATA parameter points to memory used as a temporary buffer for a new log record and must be assigned to an M or DB location.

You can assign an entire DB (derived from a PLC data type that you assign when the DB is created) or part of a DB (the specified DB element can be any data type, data type structure, PLC data type, or data array).

Structure data types are limited to a single nesting level. The total number of data elements declared should correspond to the number of columns specified in the header parameter. The maximum number of data elements you can assign is 253 (with a timestamp) or 255 (without a timestamp). This restriction keeps your record inside the 256 column limit of a Microsoft Excel sheet.

The DATA parameter can assign either retentive or non-retentive data elements in a "Standard" (compatible with S7-300/400) or "Optimized" DB type.

In order to write a Data log record you must first load the temporary DATA record with new process values and then execute the DataLogWrite instruction that saves new record values in the Datalog file.

HEADER parameter for the DataLogCreate instruction

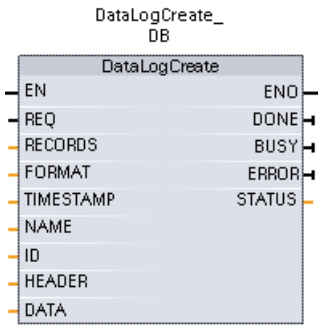
The HEADER parameter points to column header names for the top row of the data matrix encoded in the CSV file. HEADER data must be located in DB or M memory and the characters must follow standard CSV format rules with commas separating each column name. The data type may be a string, byte array, or character array. Character/byte arrays allow increased size, where strings are limited to a maximum of 255 bytes. The HEADER parameter is optional. If the HEADER is not parameterized, then no header row is created in the Data log file.

8.7.2 Program instructions that control Data logs

8.7.2.1 DataLogCreate

Table 8- 122 DataLogCreate instruction

LAD/FBD	SCL	Description

	`"DataLogCreate_DB"(`
	`req:=_bool_in_,`
	`records:=_udint_in_,`
	`format:=_uint_in_,`
	`timestamp:=_uint_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`name:=_string_inout_,`
	`ID:=_dword_inout_,`
	`header:=_variant_inout_,`
	`data:=_variant_inout_);`
		Creates and initializes a data log file. The file is created in the PLC \DataLogs directory, named by the NAME parameter, and implicitly opened for write operations. You can use the Data log instructions to programmatically store run-time process data in flash memory of the CPU.
		STEP 7 automatically creates the associated instance DB when you insert the instruction.

[2] In the SCL example, "DataLogCreate_DB" is the name of the instance DB.

Table 8- 123 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	A low to high (positive edge) signal starts the operation. (Default value: False)
RECORDS	IN	UDint	The maximum number of data records the circular data log can contain before overwriting the oldest entry: The header record is not included. Sufficient available PLC load memory must exist in order to successfully create the data log. (Default value: 1)
FORMAT	IN	UInt	Data log format: • 0: Internal format (not supported) • 1: Comma separated values "csv-eng" (Default value)
TIMESTAMP	IN	UInt	Data time stamp format: Column headers for date and time fields are not required. The time stamp uses the system time (Coordinated Universal Time - UTC) and not the local time. • 0: No time stamp • 1: Date and time stamp (Default value)
NAME	IN	Variant	Data log name: You provide the name. This variant only supports a String data type and can only be located in local, DB, or M memory. (Default value: ' ') The string reference is also used as the name of the data log file. The name characters must follow the Windows file system naming restrictions. Characters \ / : * ? " < > | and the space character are not allowed.

Parameter and type		Data type	Description
ID	In/Out	DWord	Data log numeric identifier: You store this generated value for use with other Data log instructions. The ID parameter is only used as an output with the DataLogCreate instruction. (Default value: 0) Symbolic name access for this parameter is not allowed.
HEADER	In/Out	Variant	Pointer to data log column header names for the top row of the data matrix encoded in the CSV file. (Default value: null). HEADER data must be located in DB or M memory. The characters must follow standard CSV format rules with commas separating each column name. The data type may be a string, byte array, or character array. Character/byte arrays allow increased size, where strings are limited to a maximum of 255 bytes. The HEADER parameter is optional. If the HEADER is not parameterized, then no header row is created in the Data log file.
DATA	In/Out	Variant	Pointer to the record data structure, user defined type (UDT), or array. Record data must be located in DB or M memory. The DATA parameter specifies the individual data elements (columns) of a data log record and their data type. Structure data types are limited to a single nesting level. The number of data elements declared should correspond to the number of columns specified in the header parameter. The maximum number of data elements you can assign is 253 (with a timestamp) or 255 (without a timestamp). This restriction keeps your record inside the 256 column limit of a Microsoft Excel sheet.
DONE	OUT	Bool	The DONE bit is TRUE for one scan, after the last request was completed with no error. (Default value: False)
BUSY	OUT	Bool	• 0 - No operation in progress • 1 - Operation on progress
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code (Default value: 0)

A data log file is created with a pre-determined fixed sized based on the RECORDS and DATA parameters. The data records are organized as a circular log file. New records are appended to the data log file, until the maximum number of records that is specified by the RECORDS parameter is stored. The next record written will overwrite the oldest record. Another record write operation will overwrite the next oldest data record and so on.

Note

If you want to prevent overwriting any data records, then you can use the DataLogNewFile instruction to create a new data log based on the current data log, after the current data log has stored the maximum number of records. New data records are stored in the new data log file. The old data log file and record data remains in flash memory of the CPU.

Memory resource usage:

- The data logs consume only load memory.
- There is no set limit for the total number of data logs. The size of all data logs combined is limited by the available resources of load memory. Only eight data logs may be open at one time.
- The maximum possible number for the RECORDS parameter is the limit for an UDint number (4,294,967,295). The actual limit for the RECORD parameter depends on the size of a single record, the size of other data logs, and the available resources of load memory. In addition, Microsoft Excel limits the number of rows allowed in an Excel sheet.

Note

A DataLogCreate operation extends over many program scan cycles. The actual time required for the log file creation depends on the record structure and number of records. Your program logic must monitor and catch the DataLogCreate DONE bit's transition to the TRUE state, before the new data log can be used for other data log operations.

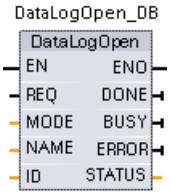
Table 8- 124 Values of ERROR and STATUS

ERROR	STATUS (W#16#....)	Description
0	0000	No error
0	7000	Call with no REQ edge: BUSY = 0, DONE = 0
0	7001	First call with REQ edge (working): BUSY = 1, DONE = 0
0	7002	N^{th} call (working): BUSY = 1, DONE = 0
1	8070	All internal instance memory is in use.
1	807F	Internal error
1	8090	Invalid file name
1	8091	Name parameter is not a String reference.
1	8093	Data log already exists.
1	8097	Requested file length exceeds file system maximum.
1	80B3	Insufficient load memory available.
1	80B4	MC (Memory Cartridge) is write protected.
1	80C1	Too many open files: No more than eight opened data log files are allowed.
1	8253	Invalid record count
1	8353	Invalid format selection
1	8453	Invalid timestamp selection
1	8B24	Invalid HEADER area assignment: For example, pointing to local memory
1	8B51	Invalid HEADER parameter data type
1	8B52	Too many HEADER parameter data elements
1	8C24	Invalid DATA area assignment: For example, pointing to local memory
1	8C51	Invalid DATA parameter data type
1	8C52	Too many DATA parameter data elements

8.7.2.2 DataLogOpen

Table 8- 125 DataLogOpen instruction

LAD / FBD	SCL	Description

	`"DataLogOpen_DB"(`
	`req:=_bool_in_,`
	`mode:=_uint_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`name:=_string_inout_,`
	`ID:=_dword_inout_);`
		Opens a pre-existing data log file. A data log must be opened before you can write new records to the log. Data logs can be opened and closed individually. A maximum of eight data logs can be open at the same time.
		STEP 7 automatically creates the associated instance DB when you insert the instruction.

[2] In the SCL example, "DataLogOpen_DB" is the name of the instance DB.

Table 8- 126 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	A low to high (positive edge) signal starts the operation. (Default value: False)
MODE	IN	UInt	Operation mode: • 0 - Append to existing data (Default value) • 1 - Clear all existing records
NAME	IN	Variant	Name of an existing data log: This variant only supports a String data type and can only be located in local, DB, or M memory. (Default value: ' ')
ID	In/Out	DWord	Numeric identifier of a data log. (Default value: 0) **Note:** Symbolic name access for this parameter is not allowed.
DONE	OUT	Bool	The DONE bit is TRUE for one scan, after the last request was completed with no error. (Default value: False)
BUSY	OUT	Bool	• 0 - No operation in progress • 1 - Operation on progress
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code (Default value: 0)

You can provide either the NAME or an ID (ID parameter as an input) of a pre-existing data log. If you provide both parameters and a valid ID does correspond to the NAME data log, then the ID is used, and the NAME ignored.

The NAME must be the name of a data log created by the DataLogCreate instruction. If only the NAME is provided and the NAME specifies a valid data log, then the corresponding ID will be returned (ID parameter as an output).

Note

General usage of data log files

- Data log files are automatically opened after the DataLogCreate and DataLogNewFile operations.
- Data log files are automatically closed after a PLC run to stop transition or a PLC power cycle.
- A Data log file must be open before a new DataLogWrite operation is possible.
- A maximum of eight data log files may be open at one time. More than eight data log files may exist, but some of them must be closed so no more than eight are open.

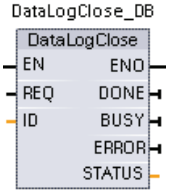
Table 8- 127 Values of ERROR and STATUS

ERROR	STATUS (W#16#)	Description
0	0000	No error
0	0002	Warning: Data log file already open by this application program
0	7000	Call with no REQ edge: BUSY = 0, DONE = 0
0	7001	First call with REQ edge (working): BUSY = 1, DONE = 0
0	7002	N^{th} call (working): BUSY = 1, DONE = 0
1	8070	All internal instance memory is in use.
1	8090	Data log definition is inconsistent with existing data log file.
1	8091	Name parameter is not a String reference.
1	8092	Data log does not exist.
1	80C0	Data log file is locked.
1	80C1	Too many open files: No more than eight opened data log files are allowed.

8.7.2.3 DataLogClose

Table 8- 128 DataLogClose instruction

LAD / FBD	SCL	Description

	`"DataLogClose_DB"(`
	`req:=_bool_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`ID:=_dword_inout_);`
		Closes an open data log file. DataLogWrite operations to a closed data log result in an error. No write operations are allowed to this data log until another DataLogOpen operation is performed.
		A transition to STOP mode will close all open data log files.
		STEP 7 automatically creates the associated instance DB when you insert the instruction.

[2] In the SCL example, "DataLogClose_DB" is the name of the instance DB.

Table 8- 129 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	A low to high (positive edge) signal starts the operation. (Default value: False)
ID	In/Out	DWord	Numeric identifier of a data log. Only used as an input for the DataLogClose instruction. (Default value: 0) **Note:** Symbolic name access for this parameter is not allowed.
DONE	OUT	Bool	The DONE bit is TRUE for one scan after the last request was completed with no error.
BUSY	OUT	Bool	• 0 - No operation in progress • 1- Operation on progress
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code (Default value: 0)

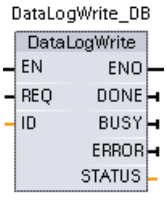
Table 8- 130 Values of ERROR and STATUS

ERROR	STATUS (W#16#)	Description
0	0000	No error
0	0001	Data log not open
0	7000	Call with no REQ edge: BUSY = 0, DONE = 0
0	7001	First call with REQ edge (working): BUSY = 1, DONE = 0
0	7002	N^{th} call (working): BUSY = 1, DONE = 0
1	8092	Data log does not exist.

8.7.2.4 DataLogWrite

Table 8- 131 DataLogWrite instruction

LAD / FBD	SCL	Description

	`"DataLogWrite_DB"(`
	`req:=_bool_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`ID:=_dword_inout_);`
		Writes a data record into the specified data log. The pre-existing target data log must be open before a DataLogWrite operation is allowed.
		STEP 7 automatically creates the associated instance DB when you insert the instruction.

[2] In the SCL example, "DataLogWrite_DB" is the name of the instance DB.

Table 8- 132 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	A low to high (positive edge) signal starts the operation. (Default value: False)
ID	In/Out	DWord	Numeric data log identifier. Only used as an input for the DataLogWrite instruction. (Default value: 0) **Note:** Symbolic name access for this parameter is not allowed.
DONE	OUT	Bool	The DONE bit is TRUE for one scan, after the last request was completed with no error.
BUSY	OUT	Bool	• 0 - No operation in progress • 1 - Operation on progress
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code (Default value: 0)

The memory address and data structure of the record buffer is configured by the DATA parameter of a DataLogCreate instruction. You must programmatically load the record buffer with current run-time process values and then execute the DataLogWrite instruction to move new record data from the buffer to the data log.

The ID parameter identifies a data log and data record configuration. The ID number is generated when a data log is created.

If there are empty records in the circular data log file, then the next available empty record will be written. If all records are full, then the oldest record will be overwritten.

CAUTION
Potential for data log data loss during a CPU power failure If there is a power failure during an incomplete DataLogWrite operation, then the data record being transferred to the data log could be lost.

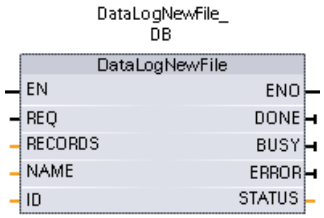
Table 8- 133 Values of ERROR and STATUS

ERROR	STATUS (W#16#)	Description
0	0000	No error
0	0001	Indicates that the data log is full: Each data log is created with a specified maximum number of records. The last record of the maximum number has been written. The next write operation will overwrite the oldest record.
0	7000	Call with no REQ edge: BUSY = 0, DONE = 0
0	7001	First call with REQ edge (working): BUSY = 1, DONE = 0
0	7002	N^{th} call (working): BUSY = 1, DONE = 0
1	8070	All internal instance memory is in use.
1	8092	Data log does not exist.
1	80B0	Data log file is not open (for explicit open mode only).

8.7.2.5 DataLogNewFile

Table 8- 134 DataLogNewFile instruction

LAD / FBD	SCL	Description

	`"DataLogNewFile_DB"(`
	`req:=_bool_in_,`
	`records:=_udint_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`name:=_DataLog_out_,`
	`ID:=_dword_inout_);`
		Allows your program to create a new data log file based upon an existing data log file.
		STEP 7 automatically creates the associated instance DB when you insert the instruction.

[2] In the SCL example, "DataLogNewFile_DB" is the name of the instance DB.

Table 8- 135 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	A low to high (positive edge) signal starts the operation. (Default value: False)
RECORDS	IN	UDInt	The maximum number of data records the circular data log can contain before overwriting the oldest entry. (Default value: 1) The header record is not included. Sufficient available CPU load memory must exist in order to successfully create the data log.
NAME	IN	Variant	Data log name: You provide the name. This variant only supports a String data type and can only be located in local, DB, or M memory. (Default value: ' ') The string reference is also used as the name of the data log file. The name characters must follow the Windows file system naming restrictions. Characters \ / : * ? " < > | and the space character are not allowed.)
ID	In/Out	DWord	Numeric data log identifier(Default value: 0): • At execution, the ID input identifies a valid data log. The new data log configuration is copied from this data log. • After execution, the ID parameter becomes an output that returns the ID of the newly created data log file. **Note:** Symbolic name access for this parameter is not allowed.
DONE	OUT	Bool	The DONE bit is TRUE for one scan, after the last request was completed with no error.
BUSY	OUT	Bool	• 0 - No operation in progress • 1 - Operation on progress
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code (Default value: 0)

You can execute the DataLogNewFile instruction when a data log becomes full or is deemed completed and you do not want to lose any data that is stored in the data log. A new empty data log file can be created based on the structure of the full Data log file. The header record will be duplicated from the original data log with the original data log properties (DATA record buffer, data format, and timestamp settings). The original Data log file is implicitly closed and the new Data log file is implicitly opened.

DataLogWrite parameter trigger: Your program must monitor the ERROR and STATUS parameters of each DataLogWrite operation. When the final record is written and a data log is full, the DataLogWrite ERROR bit = 1 and the DataLogWrite STATUS word = 1. These ERROR and STATUS values are valid for one scan only, so your monitoring logic must use ERROR = 1 as a time gate to capture the STATUS value and then test for STATUS = 1 (the data log is full).

DataLogNewFile operation: When your program logic gets the data log is full signal, this state is used to activate a DataLogNewFile operation. You must execute DataLogNewFile with the ID of an existing (usually full) and open data log, but a new unique NAME parameter. After the DataLogNewFile operation is done, a new data log ID value is returned (as an output parameter) that corresponds to the new data log name. The new data log file is implicitly opened and is ready to store new records. New DataLogWrite operations that are directed to the new data log file, must use the ID value returned by the DataLogNewFile operation.

Note

A DataLogNewFile operation extends over many program scan cycles. The actual time required for the log file creation depends on the record structure and number of records. Your program logic must monitor and catch the DataLogNewFile DONE bit's transition to the TRUE state, before the new data log can be used for other data log operations.

Table 8- 136 Values of ERROR and STATUS

ERROR	STATUS (W#16#)	Description
0	0000	No error
0	7000	Call with no REQ edge: BUSY = 0, DONE = 0
0	7001	First call with REQ edge (working): BUSY = 1, DONE = 0
0	7002	N^{th} call (working): BUSY = 1, DONE = 0
1	8070	All internal instance memory is in use.
1	8090	Invalid file name
1	8091	Name parameter is not a String reference.
1	8092	Data log does not exist.
1	8093	Data log already exists.
1	8097	Requested file length exceeds file system maximum.
1	80B3	Insufficient load memory available.
1	80B4	MC is write protected.
1	80C1	Too many open files.

8.7.3 Working with data logs

The data log files are stored as comma separated value format (*.csv) in persistent flash memory. You can view the data logs by using the PLC Web server feature or by removing the PLC memory card and inserting it in a standard PC card reader.

Viewing data logs with the PLC Web server feature

If the PLC PROFINET port and a PC are connected to a network, then you can use a PC web browser like Microsoft Internet Explorer or Mozilla Firefox to access the built-in PLC Web server. The PLC may be in run mode or stop mode when you operate the PLC Web server. If the PLC is in run mode, then your control program continues to execute while the PLC Web server is transferring log data through the network.

Web server access:

1. Enable the Web server in the Device Configuration for the target CPU (Page 474).
2. Connect your PC to the PLC through the PROFINET network (Page 474).
3. Log in to the built-in Web server (Page 476).
4. Download the recent records, all records, clear records, or delete log files with the "Data Log" standard web page (Page 486).
5. After a copy of a data log file is downloaded to your PC, you can open the .csv file with a spreadsheet application like Microsoft Excel.

Viewing data logs on a PLC memory card

If the S7-1200 CPU has a "Program" type S7-1200 memory card inserted, then you can remove the memory card and insert the card into a standard SD (Secure Digital) or MMC (MultiMediaCard) card slot on a PC or PG. The PLC is in stop mode when the memory card is removed and your control program is not executed.

Use the Windows file explorer and navigate to the \DataLog directory on the memory card. All your *.csv data log files are located in this directory.

Make a copy of the data log files and put the copies on a local drive of your PC. Then, you can use Microsoft Excel to open a local copy of a *.csv file and not the original file that is stored on the memory card.

CAUTION
You can copy, but do not modify or delete data log files on a S7-1200 memory card using a PC card reader The standard Web server data log page is the recommended tool for viewing, downloading (copying), clearing (delete the data), and deleting data log files. The Web server manages the memory card files for you and helps prevent accidental modification or deletion of data. Direct browsing of the memory card file system by the Windows Explorer has the risk that you can accidentally delete/modify data log or other system files which may corrupt a file or make the memory card unusable.

Viewing data logs from a Web browser

Even if you do not use the Web server feature, you can view data logs directly from a Web browser such as Internet Explorer or Mozilla Firefox. Simply enter the following text into the address bar of your browser using the IP address of your CPU and the actual name of the data log file you provided in STEP 7 instead of "MyDataLog":

```
http://192.168.0.1/DataLog.html?FileName=MyDataLog.csv
```

The fixed addresses of data log files also make it possible to access them through third party file collection tools.

8.7.4 Limits to the size of data log files

Data log files share PLC load memory space with the program, program data, configuration data, user-defined web pages, and PLC system data. A large program using internal load memory requires a large amount of load memory and there may be insufficient free space for data log files. In this case, you can use a "Program card" to increase the size of load memory. S7-1200 CPUs can use either internal or external load memory, but not both at once.

Refer to the memory card chapter for details about how to create a "Program" card (Page 104).

Maximum size rule for one Data log file

The maximum size of one Data log file may not exceed 25% of the load memory size (internal or external). If your application requires more Data log entries, then use the "DataLogNewFile" instruction to create a new file when all records in the first file are filled. See the table below for the maximum sizes of one Data log file.

Table 8- 137 Load memory size and maximum size for one Data log file

Data area	CPU 1211	CPU 1212	CPU 1214	Data storage
Internal load memory flash memory	1 MB (250 KB max. for one Data log file)	1 MB (250 KB max. for one Data log file)	2 MB (500 KB max. for one Data log file)	User program and program data, configuration data, Data logs, user-defined web pages, and PLC system data
External load memory Optional "Program card" flash memory cards	2 MB or 24 MB depending on the SD card size (500 KB max. for one Data log file using a 2 MB card) (6 MB max. for one Data log file using a 24 MB card)			

Determine the size of load memory free space

1. Establish an online connection between STEP 7 and the target S7-1200 PLC.
2. Download the program to which you want to add data log operations.
3. Create any optional user-defined web pages that you need. (The standard web pages that give you access to data logs are stored in PLC firmware and do not use load memory).
4. Use the Online and diagnostic tools to get the load memory size and percentage of free load memory space (Page 640).
5. Multiply the load memory size by the percentage that is free to obtain the current load memory free space.

Maximum size rule for all data logs combined

The amount of load memory free space varies during normal operations as the operating system uses and releases memory. You should limit the combined size of all data log files to one half of the available free space.

Calculate the memory requirement for a single data log record

Log data is stored as character bytes in the CSV (comma separated values) file format. The following table shows the number of bytes that are required to store each data type.

Table 8- 138 CSV file data sizes

Data type	Number of bytes (data bytes plus separator comma byte)
Bool	2
Byte	5
Word	7
DWord	12
Char	4
String	257 (Fixed size): Regardless of number of actual text characters The string text characters + automatic padding with blank characters = 254 bytes Opening and closing double quote + comma characters = 3 bytes 254 + 3 = 257 bytes
USInt	5
UInt	7
UDInt	12
SInt	5
Int	7
DInt	12
Real	16
LReal	25
Time	15
DTL	24

The DataLogCreate DATA parameter points to a structure that specifies the number of data fields and the data type of each data field for one data log record. The table above gives the bytes required in the CSV file for each data type. Multiply the number of occurrences of a given data type by the number of bytes it requires. Do this for each data type in the record and sum the number of bytes to get the total size of the data record. Add one byte for the end of line character.

Size of a data log record = summation of bytes required for all data fields + 1 (the end of line character).

Calculate the memory requirement for an entire data log file

The RECORDS parameter of the DataLogCreate instruction sets the maximum number of records in a data log file. When the data log file is created the maximum memory size is allocated.

Size of data log file = (number of bytes in one record) x (number of records).

8.7.5 Data log example program

This Data log example program does not show all the program logic necessary to get sample values from a dynamic process, but does show the key operations of the Data log instructions. The structure and number of log files that you use depends on your process control requirements.

Note

General usage of Data log files

- Data log files are automatically opened after the DataLogCreate and DataLogNew File operations.
- Data log files are automatically closed after a PLC run to stop transition or a PLC power cycle.
- A Data log file must be open before a DataLogWrite operation is possible.
- A maximum of eight data log files may be open at one time. More than eight data log files may exist, but some of them must be closed so no more than eight are open.

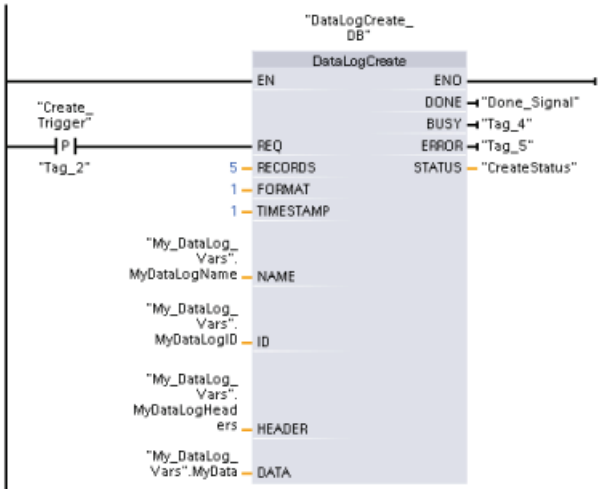
Example Data log program

Example data log names, header text, and the MyData structure are created in a data block. The three MyData variables temporarily store new sample values. The process sample values at these DB locations are transferred to a data log file by executing the DataLogWrite instruction.

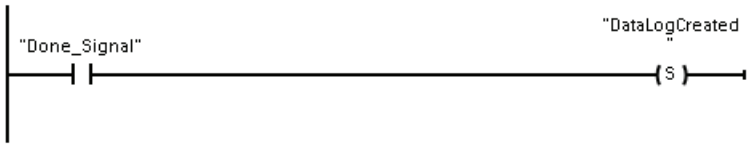
My_Datalog_Vars

	Name	Data type	Start value
1	Static		
2	MyNEWDataLogName	String	'MyNEWDataLog'
3	MyDataLogName	String	'MyDataLog'
4	MyDataLogID	DWord	0
5	MyDataLogHeaders	String	'Count,Temperature,Pressure'
6	MyData	Struct	
7	MyCount	Int	0
8	MyTermperature	Real	0.0
9	MyPressure	Real	0.0

Network 1 REQ rising edge starts the data log creation process.

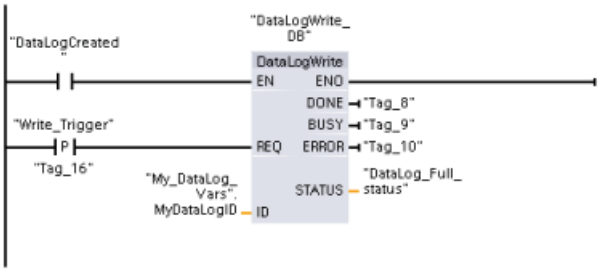


Network 2 Capture the DONE output from DataLogCreate because it is only valid for one scan.

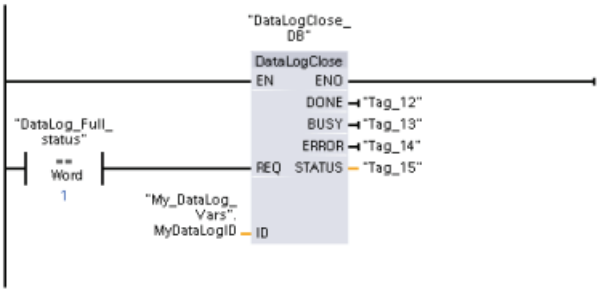


Network 3 A positive edge signal triggers when to store new process values in the MyData structure.

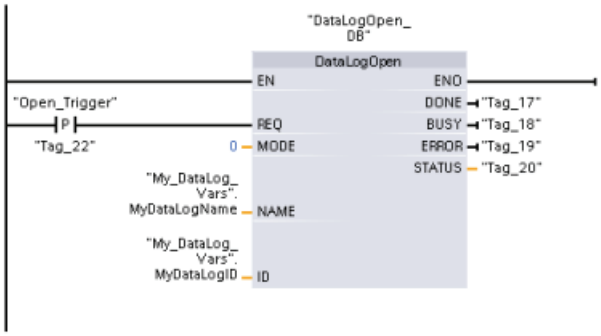
Network 4 The EN input state is based upon when the DataLogCreate operation is complete. A create operation extends over many scan cycles and must be complete before executing a write operation. The positive edge signal on the REQ input is the event that triggers an enabled write operation.



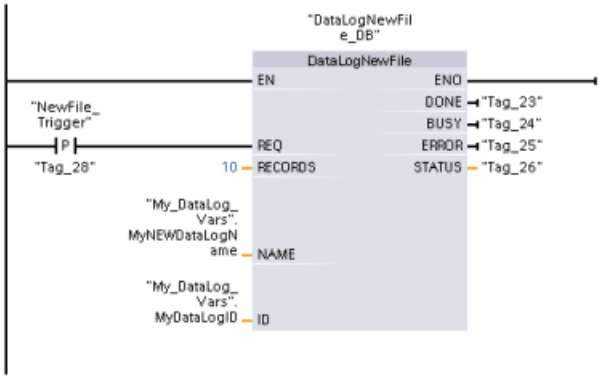
Network 5 Close the data log once the last record has been written. After executing the DataLogWrite operation that writes the last record, the log file full status is signaled when DataLogWrite STATUS output = 1.



Network 6 A positive signal edge DataLogOpen REQ input simulates the user pushing a button on an HMI that opens a data log file. If you open a Data log file that has all records filled with process data, then the next DataLogWrite operation will overwrite the oldest record. You may want to preserve the old Data log and instead create a new data log, as shown in network 7.



Network 7 The ID parameter is an IN/OUT type. First, you supply the ID value of the existing Data log whose structure you want to copy. After the DataLogNewFile operation is complete, a new and unique ID value for the new Data log is written back to the ID reference location. The required DONE bit = TRUE capture is not shown, refer to networks 1, 2, and 4 for an example of DONE bit logic.



Data log files created by the example program viewed with the S7-1200 CPU Webserver

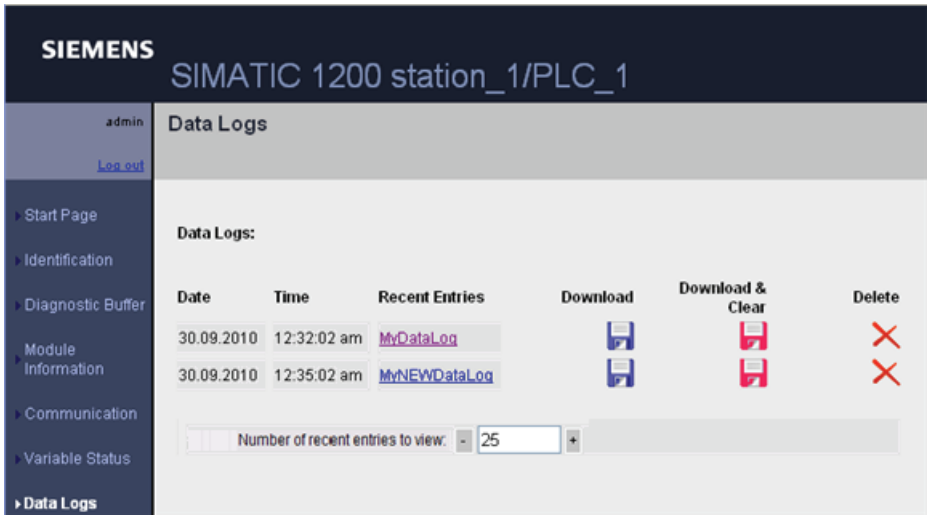


Table 8- 139 Downloaded .csv file examples viewed with Microsoft Excel

Two records written in a five record maximum file	
Five records in a Data log file with a five record maximum	
After one additional record is written to the file above which is full, the sixth write operation overwrites the oldest record one with record six. Another write operation will overwrite record two with record seven and so on.	

	A	B	C	D	E	F
1	Record	Date	Time	Count	Temperature	Pressure
2	1	9/29/2010	21:01:46	5	5.00E+00	5.00E+00
3	2	9/29/2010	21:01:47	5	5.00E+00	5.00E+00
4	//END					
5						

	A	B	C	D	E	F
1	Record	Date	Time	Count	Temperature	Pressure
2	1	9/30/2010	20:26:56	1	9.86E+01	3.52E+01
3	2	9/30/2010	20:28:43	2	1.00E+02	3.73E+01
4	3	9/30/2010	20:29:03	3	9.99E+01	3.68E+01
5	4	9/30/2010	20:29:21	4	9.95E+01	3.64E+01
6	5	9/30/2010	20:30:19	5	9.92E+01	3.74E+01
7						

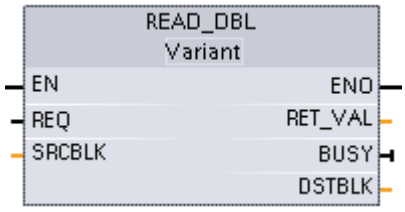
	A	B	C	D	E	F
1	Record	Date	Time	Count	Temperature	Pressure
2	6	9/30/2010	20:32:03	6	9.86E+01	3.58E+01
3	2	9/30/2010	20:28:43	2	1.00E+02	3.73E+01
4	3	9/30/2010	20:29:03	3	9.99E+01	3.68E+01
5	4	9/30/2010	20:29:21	4	9.95E+01	3.64E+01
6	5	9/30/2010	20:30:19	5	9.92E+01	3.74E+01
7						

8.8 Data block control

8.8.1 READ_DBL, WRIT_DBL (Read from or write to a DB in load memory)

Table 8- 140 READ_DBL and WRIT_DBL instructions

LAD / FBD		Description

	`READ_DBL(`
	`req:=_bool_in_,`
	`srcblk:=_variant_in_,`
	`busy=>_bool_out_,`
	`dstblk=>_variant_out_);`
		Copies DB start values or part of the values, from load memory to a target DB in the work memory.
		The content of load memory is not changed during the copy process.

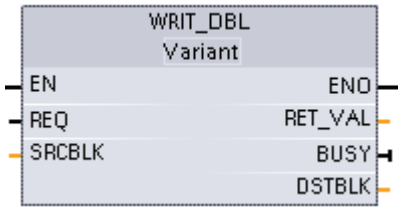
	`WRIT_DBL(`
	`req:=_bool_in_,`
	`srcblk:=_variant_in_,`
	`busy=>_bool_out_,`
	`dstblk=>_variant_out_);`
		Copies DB current values or part of the values from work memory to a target DB in load memory.
		The content of work memory is not changed during the copy process.

Table 8- 141 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	BOOL	A high signal starts the operation, if BUSY = 0.
SRCBLK	IN	VARIANT	READ_DBL: Pointer to the source data block in load memory WRIT_DBL: Pointer to the source data block in work memory
RET_VAL	OUT	INT	Execution condition code
BUSY	OUT	BOOL	BUSY = 1 signals that the reading/writing process is not complete.
DSTBLK	OUT	VARIANT	READ_DBL: Pointer to the destination data block in work memory WRIT_DBL: Pointer to the destination data block in load memory

Typically, a DB is stored in both load memory (flash) and work memory (RAM). The start values (initial values) are always stored in load memory, and the current values are always stored in work memory. READ_DBL can be used to copy a set of start values from load memory to the current values of a DB in work memory that is referenced by your program. You can use WRIT_DBL to update the start values stored in internal load memory or memory card from current values in work memory.

Note

Avoid excessive WRIT_DBL flash memory write operations

The WRIT_DBL instruction performs write operations in flash memory (internal load memory or memory card). WRIT_DBL should be used for infrequent updates like a production process changes.

The data blocks used by READ_DBL and WRIT_DBL must have been previously created by STEP 7 before you can use these instructions. If the source DB is created as a "standard" type then the destination DB must also be the "standard" type. If the source data block is created as an "optimized" type then the destination data block must also be the "optimized" type.

If the DBs are standard, then you can specify either a tag name or a P# value. The P# value allows you to specify and copy any number of elements of the specified size (Byte, Word, or DWord). Thus, you can copy part or all of a DB. If the DBs are optimized, you can only specify a tag name; you cannot use the P# operator. If you specify a tag name for either standard or optimized DBs (or for other work-memory types), then whatever is referenced by this tag name is copied. This could be a user-defined type, an array, or a basic element. Type Struct can only be used by these instructions if the DB is standard, not optimized. You must use a user-defined type (UDT) if it is a structure in optimized memory. Only a user-defined type ensures that the "data types" are exactly the same for both the source and destination structures.

Note

Using a structure (data type Struct) in an "optimized" DB

When using a Struct data type with "optimzed" DBs, you must first create a user-defined data type (UDT) for the Struct. You then configure both the source and destination DBs with the UDT. The UDT ensures that the data types within the Struct remain consistent for both DBs.

For "standard" DBs, you use the Struct without creating a UDT.

READ_DBL and WRIT_DBL execute asynchronously to the cyclic program scan. The processing extends over multiple READ_DBL and WRIT_DBL calls. You start the DB transfer job by calling with REQ = 1 and then monitor the BUSY and RET_VAL outputs to determine when the data transfer is complete and correct.

To ensure data consistency, do not modify the destination area during the processing of READ_DBL or the source area during the processing of WRIT_DBL (that is, as long as the BUSY parameter is TRUE).

SRCBLK and DSTBLK parameter restrictions:

- A data block must have been previously created before it can be referenced.
- The length of a VARIANT pointer of type BOOL must be divisible by 8.
- The length of a VARIANT pointer of type STRING must be the same in the source and destination pointers.

Recipes and machine setup information

You can use the READ_DBL and WRIT_DBL instructions to manage recipes or machine setup information. This essentially becomes another method of achieving retentive data for values that do not change often, although you would want to limit the number of writes to prevent wearing out the flash prematurely. This effectively allows you to increase the amount of retentive memory beyond that supported for the normal power-down retentive data, at least for values that do not change often. You could save recipe information or machine-setup information from work memory to load memory using the WRIT_DBL instruction, and you could retrieve such information from load memory into work memory using the READ_DBL instruction.

Table 8- 142 Condition codes

RET_VAL (W#16#...)	Description
0000	No error
0081	Warning: that the source area is smaller than the destination area. The source data is copied completely with the extra bytes in the destination area unchanged.
7000	Call with REQ = 0: BUSY = 0
7001	First call with REQ = 1 (working): BUSY = 1
7002	N^{th} call (working): BUSY = 1
8051	Data block type error
8081	The source area is larger than the destination area. The destination area is completely filled and the remaining bytes of the source are ignored.
8251	Source data block type error
82B1	Missing source data block
82C0	The source DB is being edited by another statement or a communication function.
8551	Destination data block type error
85B1	Missing destination data block
85C0	The destination DB is being edited by another statement or a communication function.
80C3	More than 50 READ_DBL or 50 WRIT_DBL statements are currently queued for execution.

8.9 Common error codes for the "Extended" instructions

Table 8- 143 Common condition codes for the extended instructions

Condition code (W#16#....)[1]	Description
8022	Area too small for input
8023	Area too small for output
8024	Illegal input area
8025	Illegal output area
8028	Illegal input bit assignment
8029	Illegal output bit assignment
8030	Output area is a read-only DB.
803A	DB does not exist.

[1] If one of these errors occurs when a code block is executed the the CPU goes to STOP mode, unless you use the GetError or GetErrorID instructions within that code block and create a programmed reaction to the error.

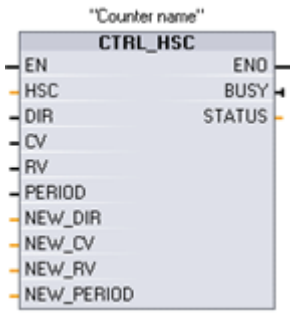
Technology instructions

9

9.1 High-speed counter

Table 9- 1 CTRL_HSC instruction

LAD / FBD	SCL	Description

	`"CTRL_HSC_0_DB" (`
	`hsc:=_hw_hsc_in_,`
	`dir:=_bool_in_,`
	`cv:=_bool_in_,`
	`rv:=_bool_in_,`
	`period:=_bool_in_,`
	`new_dir:=_int_in_,`
	`new_cv:=_int_in_,`
	`new_rv:=_dint_in_,`
	`new_period:=_int_in_,`
	`busy:=_bool_out_,`
	`status:=_word_out_);`
		Each CTRL_HSC instruction uses a structure stored in a DB to maintain data. You assign the DB when the CTRL_HSC instruction is placed in the editor.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "CTRL_HSC_0_DB" is the name of the instance DB.

Table 9- 2 Data types for the parameters

Parameter and type		Data type	Description
HSC	IN	HW_HSC	HSC identifier
DIR[1, 2]	IN	Bool	1 = Request new direction
CV[1]	IN	Bool	1 = Request to set new counter value
RV[1]	IN	Bool	1= Request to set new reference value
PERIOD[1]	IN	Bool	1 = Request to set new period value (only for frequency measurement mode)
NEW_DIR	IN	Int	New direction: 1= forward, -1= backward
NEW_CV	IN	DInt	New counter value
NEW_RV	IN	DInt	New reference value
NEW_PERIOD	IN	Int	New period value in seconds: 0.01, 0.1, or 1 (only for frequency measurement mode)
BUSY[3]	OUT	Bool	Function is busy
STATUS	OUT	Word	Execution condition code

1 If an update of a parameter value is not requested, then the corresponding input values are ignored.

2 The DIR parameter is only valid if the configured counting direction is set to "User program (internal direction control)". You determine how to use this parameter in the HSC device configuration.

3 For an HSC on the CPU or on the SB, the BUSY parameter always has a value of 0.

You configure the parameters for each HSC in the device configuration for the CPU: counting mode, I/O connections, interrupt assignment, and operation as a high-speed counter or as a device to measure pulse frequency.

Some of the parameters for the HSC can be modified by your user program to provide program control of the counting process:

- Set the counting direction to a NEW_DIR value
- Set the current count value to a NEW_CV value
- Set the reference value to a NEW_RV value
- Set the period value (for frequency measurement mode) to a NEW_PERIOD value

If the following Boolean flag values are set to 1 when the CTRL_HSC instruction is executed, the corresponding NEW_xxx value is loaded to the counter. Multiple requests (more than one flag is set at the same time) are processed in a single execution of the CTRL_HSC instruction.

- DIR = 1 is a request to load a NEW_DIR value, 0 = no change
- CV = 1 is a request to load a NEW_CV value, 0 = no change
- RV = 1 is a request to load a NEW_RV value, 0 = no change
- PERIOD = 1 is a request to load a NEW_PERIOD value, 0 = no change

The CTRL_HSC instruction is typically placed in a hardware interrupt OB that is executed when the counter hardware interrupt event is triggered. For example, if a CV=RV event triggers the counter interrupt, then a hardware interrupt OB code block executes the CTRL_HSC instruction and can change the reference value by loading a NEW_RV value.

The current count value is not available in the CTRL_HSC parameters. The process image address that stores the current count value is assigned during the hardware configuration of the high-speed counter. You may use program logic to directly read the count value. The value returned to your program will be a correct count for the instant in which the counter was read. The counter will continue to count high-speed events. Therefore, the actual count value could change before your program completes a process using an old count value.

Condition codes: In the case of an error, ENO is set to 0, and the STATUS output contains a condition code.

Table 9- 3 STATUS values (W#16#)

STATUS	Description
0	No error
80A1	HSC identifier does not address a HSC
80B1	Illegal value in NEW_DIR
80B2	Illegal value in NEW_CV
80B3	Illegal value in NEW_RV
80B4	Illegal value in NEW_PERIOD
80C0	Multiple access to the high-speed counter

9.1.1 Operation of the high-speed counter

The high-speed counter (HSC) counts events that occur faster than the OB execution rate. If the events to be counted occur within the execution rate of the OB, you can use CTU, CTD, or CTUD counter instructions. If the events occur faster than the OB execution rate, then use the HSC. The CTRL_HSC instruction allows your user program to programmatically change some of the HSC parameters.

For example: You can use the HSC as an input for an incremental shaft encoder. The shaft encoder provides a specified number of counts per revolution and a reset pulse that occurs once per revolution. The clock(s) and the reset pulse from the shaft encoder provide the inputs to the HSC.

The HSC is loaded with the first of several presets, and the outputs are activated for the time period where the current count is less than the current preset. The HSC provides an interrupt when the current count is equal to preset, when reset occurs, and also when there is a direction change.

As each current-count-value-equals-preset-value interrupt event occurs, a new preset is loaded and the next state for the outputs is set. When the reset interrupt event occurs, the first preset and the first output states are set, and the cycle is repeated.

Since the interrupts occur at a much lower rate than the counting rate of the HSC, precise control of high-speed operations can be implemented with relatively minor impact to the scan cycle of the CPU. The method of interrupt attachment allows each load of a new preset to be performed in a separate interrupt routine for easy state control. (Alternatively, all interrupt events can be processed in a single interrupt routine.)

Table 9- 4 Maximum frequency (KHz)

HSC		Single phase	Two phase and AB quadrature
HSC1	CPU	100 KHz	80 KHz
	High-speed SB	200 KHz	160 KHz
	SB	30 KHz	20 KHz
HSC2	CPU	100 KHz	80 KHz
	High-speed SB	200 KHz	160 KHz
	SB	30 KHz	20 KHz
HSC3	CPU	100 KHz	80 KHz
HSC4	CPU	30 KHz	20 KHz
HSC5	CPU	30 KHz	20 KHz
	High-speed SB	200 KHz	160 KHz
	SB	30 KHz	20 KHz
HSC6	CPU	30 KHz	20 KHz
	High-speed SB	200 KHz	160 KHz
	SB	30 KHz	20 KHz

Selecting the functionality for the HSC

All HSCs function the same way for the same counter mode of operation. There are four basic types of HSC:

- Single-phase counter with internal direction control
- Single-phase counter with external direction control
- Two-phase counter with 2 clock inputs
- A/B phase quadrature counter

You can use each HSC type with or without a reset input. When you activate the reset input (with some restrictions, see the following table), the current value is cleared and held clear until you deactivate the reset input.

- Frequency function: Some HSC modes allow the HSC to be configured (Type of counting) to report the frequency instead of a current count of pulses. Three different frequency measuring periods are available: 0.01, 0.1, or 1.0 seconds.

  The frequency measuring period determines how often the HSC calculates and reports a new frequency value. The reported frequency is an average value determined by the total number of counts in the last measuring period. If the frequency is rapidly changing, the reported value will be an intermediate between the highest and lowest frequency occurring during the measuring period. The frequency is always reported in Hertz (pulses per second) regardless of the frequency-measuring-period setting.

- Counter modes and inputs: The following table shows the inputs used for the clock, direction control, and reset functions associated with the HSC.

  The same input cannot be used for two different functions, but any input not being used by the present mode of its HSC can be used for another purpose. For example, if HSC1 is in a mode that uses built-in inputs but does not use the external reset (I0.3), then I0.3 can be used for edge interrupts or for HSC2.

Table 9- 5 Counting modes for HSC

Type	Input 1	Input 2	Input 3	Function
Single-phase counter with internal direction control	Clock	(Optional: direction)	-	Count or frequency
			Reset	Count
Single-phase counter with external direction control	Clock	Direction	-	Count or frequency
			Reset	Count
Two-phase counter with 2 clock inputs	Clock up	Clock down	-	Count or frequency
			Reset	Count
A/B-phase quadrature counter	Phase A	Phase B	-	Count or frequency
			Reset[1]	Count

[1] For an encoder: Phase Z, Home

Input addresses for the HSC

Note

The digital I/O points used by high-speed counter devices are assigned during device configuration. When digital I/O point addresses are assigned to these devices, the values of the assigned I/O point addresses cannot be modified by the force function in a watch table.

When you configure the CPU, you have the option to enable and configure each HSC. The CPU automatically assigns the input addresses for each HSC according to its configuration. (Some of the HSCs allow you to select whether to use either the on-board inputs of the CPU or the inputs of an SB.)

NOTICE

As shown in the following tables, the default assignments for the optional signals for the different HSCs overlap. For example, the optional external reset for HSC 1 uses the same input as one of the inputs for HSC 2.

Always ensure that you have configured your HSCs so that any one input is **not** being used by two HSCs.

The following table shows the HSC input assignments for both the on-board I/O of the CPU 1211C and an SB. (If the SB has only 2 inputs, only 4.0 and 4.1 inputs are available.)

- For single-phase: C is the Clock input, [d] is the optional direction input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)
- For two-phase: CU is the Clock Up input, CD is the Clock Down input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)
- For AB-phase quadrature: A is the Clock A input, B is the Clock B input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)

Table 9- 6 HSC input assignments for CPU 1211C

HSC		CPU on-board input (0.x)						SB input (default 4.x) [3]			
		0	1	2	3	4	5	0	1	2	3
HSC 1 [1]	1-phase	C	[d]		[R]			C	[d]		[R]
	2-phase	CU	CD		[R]			CU	CD		[R]
	AB-phase	A	B		[R]			A	B		[R]
HSC 2 [1]	1-phase		[R]	C	[d]				[R]	C	[d]
	2-phase		[R]	CU	CD				[R]	CU	CD
	AB-phase		[R]	A	B				[R]	A	B
HSC 3	1-phase					C	[d]				
	2-phase					CU	CD				
	AB-phase					A	B				
HSC 5 [2]	1-phase							C	[d]		[R]
	2-phase							CU	CD		[R]
	AB-phase							A	B		[R]

HSC		CPU on-board input (0.x)						SB input (default 4.x) [3]			
		0	1	2	3	4	5	0	1	2	3
HSC 6 [2]	1-phase								[R]	C	[d]
	2-phase								[R]	CU	CD
	AB-phase								[R]	A	B

[1] HSC 1 and HSC 2 can be configured for either the on-board inputs or for an SB.

[2] HSC 5 and HSC 6 are available only with an SB. HSC 6 is available only with a 4-input SB.

[3] An SB with only 2 digital inputs provides only the 4.0 and 4.1 inputs.

The following table shows the HSC input assignments for both the on-board I/O of the CPU 1212C and an SB. (If the SB has only 2 inputs, only 4.0 and 4.1 inputs are available.)

- For single-phase: C is the Clock input, [d] is the optional direction input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)
- For two-phase: CU is the Clock Up input, CD is the Clock Down input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)
- For AB-phase quadrature: A is the Clock A input, B is the Clock B input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)

Table 9- 7 HSC input assignments for CPU 1212C

HSC		CPU on-board input (0.x)								SB input (4.x) [3]			
		0	1	2	3	4	5	6	7	0	1	2	3
HSC 1 [1]	1-phase	C	[d]		[R]					C	[d]		[R]
	2-phase	CU	CD		[R]					CU	CD		[R]
	AB-phase	A	B		[R]					A	B		[R]
HSC 2 [1]	1-phase		[R]	C	[d]						[R]	C	[d]
	2-phase		[R]	CU	CD						[R]	CU	CD
	AB-phase		[R]	A	B						[R]	A	B
HSC 3	1-phase					C	[d]		[R]				
	2-phase					CU	CD		[R]				
	AB-phase					A	B		[R]				
HSC 4	1-phase						[R]	C	[d]				
	2-phase						[R]	CU	CD				
	AB-phase						[R]	A	B				
HSC 5 [2]	1-phase									C	[d]		[R]
	2-phase									CU	CD		[R]
	AB-phase									A	B		[R]
HSC 6 [2]	1-phase										[R]	C	[d]
	2-phase										[R]	CU	CD
	AB-phase										[R]	A	B

[1] HSC 1 and HSC 2 can be configured for either the on-board inputs or for an SB.

[2] HSC 5 and HSC 6 are available only with an SB. HSC 6 is available only with a 4-input SB.

[3] An SB with only 2 digital inputs provides only the 4.0 and 4.1 inputs.

The following two tables show the HSC input assignments for the on-board I/O of the CPU 1214C and for an optional SB, if installed.

- For single-phase: C is the Clock input, [d] is the optional direction input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)
- For two-phase: CU is the Clock Up input, CD is the Clock Down input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)
- For AB-phase quadrature: A is the Clock A input, B is the Clock B input, and [R] is an optional external reset input. (Reset is available only for "Counting" mode.)

Table 9- 8 HSC input assignments for CPU 1214C (on-board inputs only)

HSC		Digital input 0 (default: 0.x)								Digital input 1 (default: 1.x)					
		0	1	2	3	4	5	6	7	0	1	2	3	4	5
HSC 1[1]	1-phase	C	[d]		[R]										
	2-phase	CU	CD		[R]										
	AB-phase	A	B		[R]										
HSC 2[1]	1-phase		[R]	C	[d]										
	2-phase		[R]	CU	CD										
	AB-phase		[R]	A	B										
HSC 3	1-phase					C	[d]		[R]						
	2-phase					CU	CD		[R]						
	AB-phase					A	B		[R]						
HSC 4	1-phase						[R]	C	[d]						
	2-phase						[R]	CU	CD						
	AB-phase						[R]	A	B						
HSC 5[1]	1-phase									C	[d]	[R]			
	2-phase									CU	CD	[R]			
	AB-phase									A	B	[R]			
HSC 6[1]	1-phase												C	[d]	[R]
	2-phase												CU	CD	[R]
	AB-phase												A	B	[R]

[1] HSC 1, HSC 2, HSC 5 and HSC 6 can be configured for either the on-board inputs or for an SB.

Table 9- 9 HSC input assignments for SBs

HSC [1]		SB inputs (default: 4.x) [2]			
		0	1	2	3
HSC 1	1-phase	C	[d]		[R]
	2-phase	CU	CD		[R]
	AB-phase	A	B		[R]
HSC 2	1-phase		[R]	C	[d]
	2-phase		[R]	CU	CD
	AB-phase		[R]	A	B

HSC [1]		SB inputs (default: 4.x) [2]			
		0	1	2	3
HSC 5	1-phase	C	[d]		[R]
	2-phase	CU	CD		[R]
	AB-phase	A	B		[R]
HSC 6	1-phase		[R]	C	[d]
	2-phase		[R]	CU	CD
	AB-phase		[R]	A	B

1 For CPU 1214C: HSC 1, HSC 2, HSC 5 and HSC 6 can be configured for either the on-board inputs or for an SB.

2 An SB with only 2 digital inputs provides only the 4.0 and 4.1 inputs.

Accessing the current value for the HSC

Note

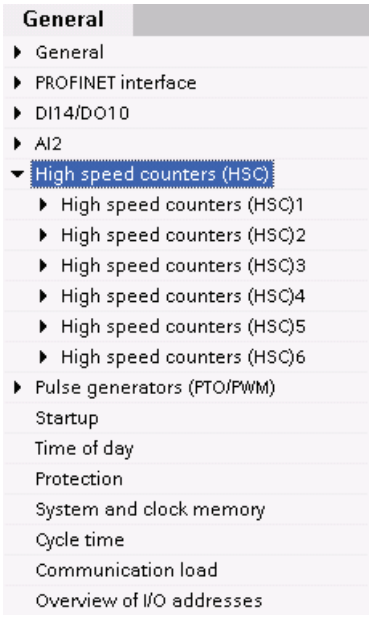
When you enable a pulse generator for use as a PTO, a corresponding HSC is assigned to this PTO. HSC1 is assigned for PTO1, and HSC2 is assigned for PTO2. The assigned HSC belongs completely to the PTO channel, and the ordinary output of the HSC is disabled. The HSC value is only used for the internal functionality. You cannot monitor the current value (for example, in ID1000) when pulses are occurring.

The CPU stores the current value of each HSC in an input (I) address. The following table shows the default addresses assigned to the current value for each HSC. You can change the I address for the current value by modifying the properties of the CPU in the Device Configuration.

Table 9- 10 Current value of the HSC

HSC	Data type	Default address
HSC1	DInt	ID1000
HSC2	DInt	ID1004
HSC3	DInt	ID1008
HSC4	DInt	ID1012
HSC5	DInt	ID1016
HSC6	DInt	ID1020

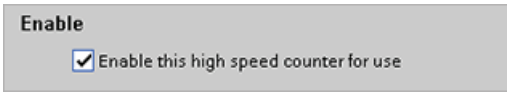
9.1.2 Configuration of the HSC



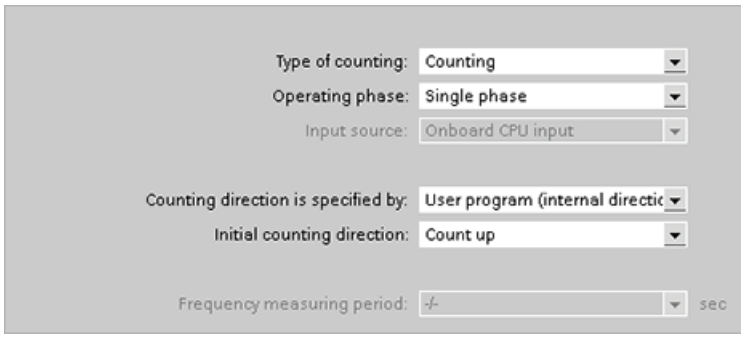
The CPU allows you to configure up to 6 high-speed counters. You edit the "Properties" of the CPU to configure the parameters of each individual HSC.

Use the CTRL_HSC instruction in your user program to control the operation of the HSC.

Enable the specific HSC by selecting the "Enable" option for that HSC.



After enabling the HSC, configure the other parameters, such as counter function, initial values, reset options and interrupt events.



For information about configuring the HSC, refer to the section on configuring the CPU (Page 114).

9.2 PID control

STEP 7 provides the following PID instructions for the S7-1200 CPU:

- The PID_Compact instruction is used to control technical processes with continuous input- and output variables.
- The PID_3Step instruction is used to control motor-actuated devices, such as valves that require discrete signals for open- and close actuation.

Both PID instructions (PID_3Step and PID_Compact) can calculate the P-, I-, and D-components during startup (if configured for "pretuning"). You can also configure the instruction for "fine tuning" to allow you to optimize the parameters. You do not need to manually determine the parameters.

Note

Execute the PID instruction at constant intervals of the sampling time (preferably in a cyclic OB).

Because the PID loop needs a certain time to respond to changes of the control value, do not calculate the output value in every cycle. Do not execute the PID instruction in the main program cycle OB (such as OB 1).

The sampling time of the PID algorithm represents the time between two calculations of the output value (control value). The output value is calculated during self-tuning and rounded to a multiple of the cycle time. All other functions of PID instruction are executed at every call.

PID algorithm

The PID (Proportional/Integral/Derivative) controller measures the time interval between two calls and then evaluates the results for monitoring the sampling time. A mean value of the sampling time is generated at each mode changeover and during initial startup. This value is used as reference for the monitoring function and is used for calculation. Monitoring includes the current measuring time between two calls and the mean value of the defined controller sampling time.

The output value for the PID controller consists of three components:

- P (proportional): When calculated with the "P" component, the output value is proportional to the difference between the setpoint and the process value (input value).
- I (integral): When calculated with the "I" component, the output value increases in proportion to the duration of the difference between the setpoint and the process value (input value) to finally correct the difference.
- D (derivative): When calculated with the "D" component, the output value increases as a function of the increasing rate of change of the difference between the setpoint and the process value (input value). The output value is corrected to the setpoint as quickly as possible.

The PID controller uses the following formula to calculate the output value for the PID_Compact instruction.

$$y = K_p \left[(b \cdot w - x) + \frac{1}{T_I \cdot s} (w - x) + \frac{T_D \cdot s}{a \cdot T_D \cdot s + 1} (c \cdot w - x) \right]$$

y	Output value	x	Process value
w	Setpoint value	s	Laplace operator
K_p	Proportional gain (P component)	a	Derivative delay coefficient (D component)
T_1	Integral action time (I component)	b	Proportional action weighting (P component)
T_D	Derivative action time (D component)	c	Derivative action weighting (D component)

The PID controller uses the following formula to calculate the output value for the PID_3Step instruction.

$$\Delta y = K_p \cdot s \cdot \left[(b \cdot w - x) + \frac{1}{T_I \cdot s} (w - x) + \frac{T_D \cdot s}{a \cdot T_D \cdot s + 1} (c \cdot w - x) \right]$$

y	Output value	x	Process value
w	Setpoint value	s	Laplace operator
K_p	Proportional gain (P component)	a	Derivative delay coefficient (D component)
T_1	Integral action time (I component)	b	Proportional action weighting (P component)
T_D	Derivative action time (D component)	c	Derivative action weighting (D component)

9.2.1 Inserting the PID instruction and technological object

STEP 7 provides two instructions for PID control:

- The PID_Compact instruction and its associated technological object provide a universal PID controller with tuning. The technological object contains all of the settings for the control loop.
- The PID_3Step instruction and its associated technological object provide a PID controller with specific settings for motor-activated valves. The technological object contains all of the settings for the control loop. The PID_3Step controller provides two additional Boolean outputs.

After creating the technological object, you must configure the parameters (Page 337). You also adjust the autotuning parameters ("pretuning" during startup or manual "fine tuning") to commission the operation of the PID controller (Page 340).

Table 9- 11 Inserting the PID instruction and the technological object

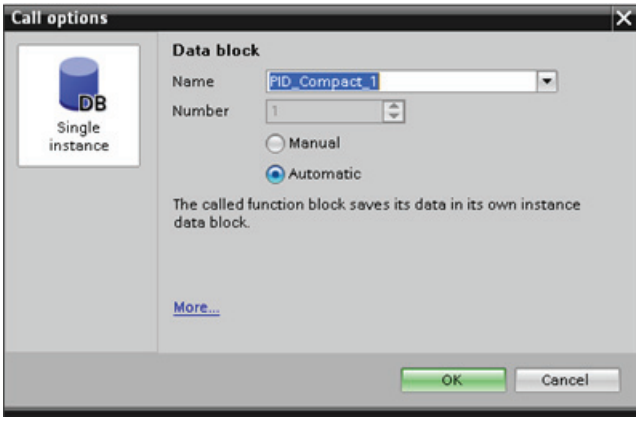
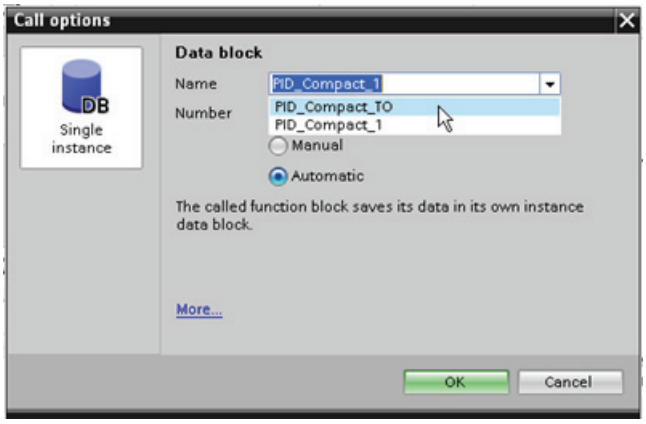
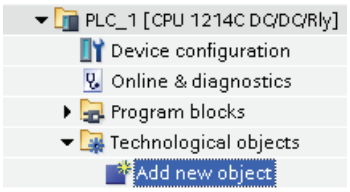
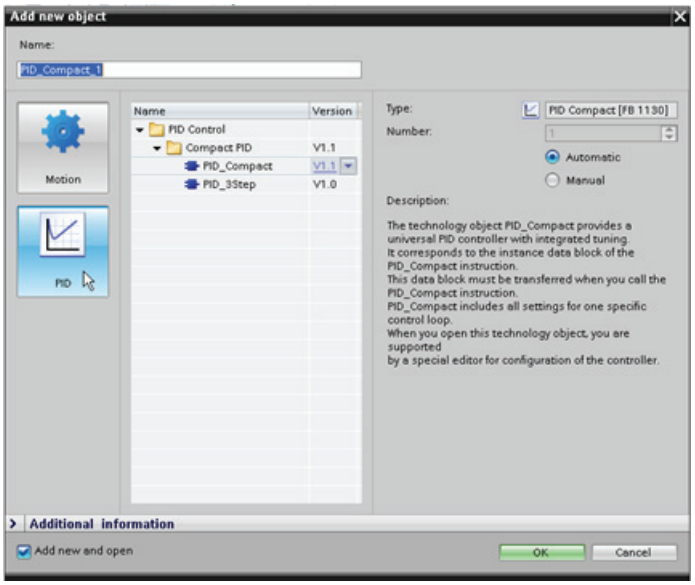
When you insert a PID instruction into your user program, STEP 7 automatically creates a technology object and an instance DB for the instruction. The instance DB contains all of the parameters that are used by the PID instruction. Each PID instruction must have its own unique instance DB to operate properly. After inserting the PID instruction and creating the technological object and instance DB, you configure the parameters for the technological object (Page 337).	

Table 9- 12 (Optional) Creating a technological object from the project navigator

You can also create technological objects for your project **before** inserting the PID instruction. By creating the technological object before inserting a PID instruction into your user program, you can then select the technological object when you insert the PID instruction.	
To create a technological object, double-click the "Add new object" icon in the project navigator.	
Click the "Control" icon and select the technological object for the type of PID controller (PID_Compact or PID_3Step). You can create an optional name for the technological object. Click "OK" to create the technological object.	

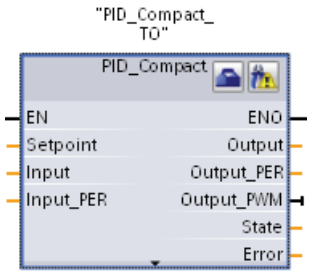
9.2.2 PID_Compact instruction

The PID controller uses the following formula to calculate the output value for the PID_Compact instruction.

$$y = K_p \left[(b \cdot w - x) + \frac{1}{T_I \cdot s}(w - x) + \frac{T_D \cdot s}{a \cdot T_D \cdot s + 1}(c \cdot w - x) \right]$$

y	Output value	x	Process value
w	Setpoint value	s	Laplace operator
K_p	Proportional gain (P component)	a	Derivative delay coefficient (D component)
T_1	Integral action time (I component)	b	Proportional action weighting (P component)
T_D	Derivative action time (D component)	c	Derivative action weighting (D component)

Table 9- 13 PID_Compact instruction

LAD / FBD	SCL	Description
	"PID_Compact_1"(Setpoint:=_real_in_, Input:=_real_in_, Input_PER:=_word_in_, ManualEnable:=_bool_in_, ManualValue:=_real_in_, Reset:=_bool_in_, ScaledInput=>_real_out_, Output=>_real_out_, Output_PER=>_word_out_, Output_PWM=>_bool_out_, SetpointLimit_H=>_bool_out_, SetpointLimit_L=>_bool_out_, InputWarning_H=>_bool_out_, InputWarning_L=>_bool_out_, State=>_int_out_, Error=>_dword_out_);	PID_Compact provides a PID controller with self-tuning for automatic and manual mode. PID_Compact is a PIDT1 controller with anti-windup and weighting of the P- and D-component.

[1] STEP 7 automatically creates the technological object and instance DB when you insert the instruction. The instance DB contains the parameters of the technological object.

[2] In the SCL example, "PID_Compact_1" is the name of the instance DB.

Table 9- 14 Data types for the parameters

Parameter and type		Data type	Description
Setpoint	IN	Real	Setpoint of the PID controller in automatic mode. Default value: 0.0
Input	IN	Real	Process value. Default value: 0.0 You must also set sPid_Cmpt.b_Input_PER_On = FALSE.

Parameter and type		Data type	Description
Input_PER	IN	Word	Analog process value (optional). Default value: W#16#0 You must also set sPid_Cmpt.b_Input_PER_On = TRUE.
ManualEnable	IN	Bool	Enables or disables the manual operation mode. Default value: FALSE • On the edge of the change from FALSE to TRUE, the PID controller switches to manual mode, State = 4, and sRet.i_Mode remains unchanged. • On the edge of the change from TRUE to FALSE, the PID controller switches to the last active operating mode and State = sRet.i_Mode.
ManualValue	IN	Real	Process value for manual operation. Default value: 0.0
Reset	IN	Bool	Restarts the controller. Default value: FALSE If Reset = TRUE, the following applies: • Inactive operating mode • Input value = 0 • Integral part of the process value = 0 • Intermediate values of the system are reset (PIDParameter is retained)
ScaledInput	OUT	Real	Scaled process value. Default value: 0.0
Output[1]	OUT	Real	Output value. Default value: 0.0
Output_PER[1]	OUT	Word	Analog output value. Default value: W#16#0
Output_PWM[1]	OUT	Bool	Output value for pulse width modulation. Default value: FALSE
SetpointLimit_H	OUT	Bool	Setpoint high limit. Default value: FALSE If SetpointLimit_H = TRUE, the absolute upper limit of the setpoint is reached. Default value: FALSE
SetpointLimit_L	OUT	Bool	Setpoint low limit. Default value: FALSE If SetpointLimit_L = TRUE, the absolute lower limit of the setpoint is reached. Default value: FALSE
InputWarning_H	OUT	Bool	If InputWarning_H = TRUE, the process value reached or exceeded the upper warning limit. Default value: FALSE
InputWarning_L	OUT	Bool	If InputWarning_L = TRUE, the process value reached the lower warning limit. Default value: FALSE
State	OUT	Int	Current operating mode of the PID controller. Default value: 0 Use sRet.i_Mode to change the mode. • State = 0: Inactive • State = 1: Pretuning • State = 2: Manual fine tuning • State = 3: Automatic mode • State = 4: Manual mode
Error	OUT	DWord	Error message Default value: DW#16#0000 (no error)

[1] The outputs of the Output, Output_PER, and Output_PWM parameters can be used in parallel.

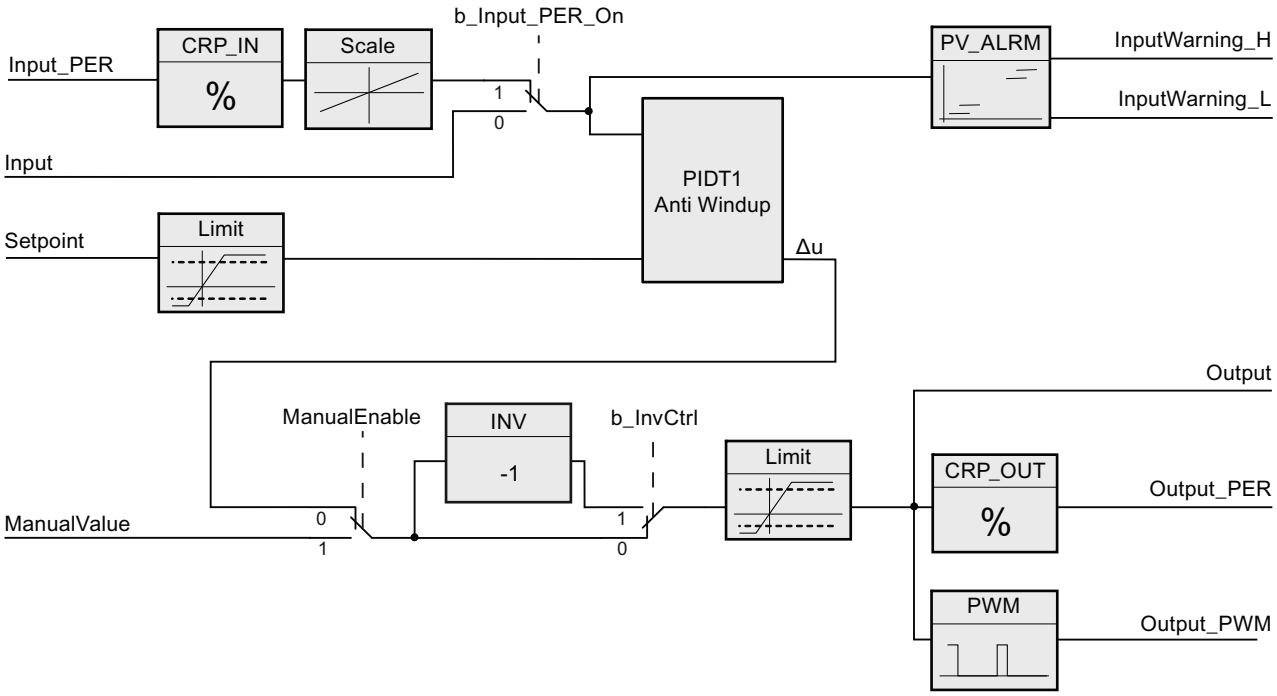


Figure 9-1 Operation of the PID_Compact controller

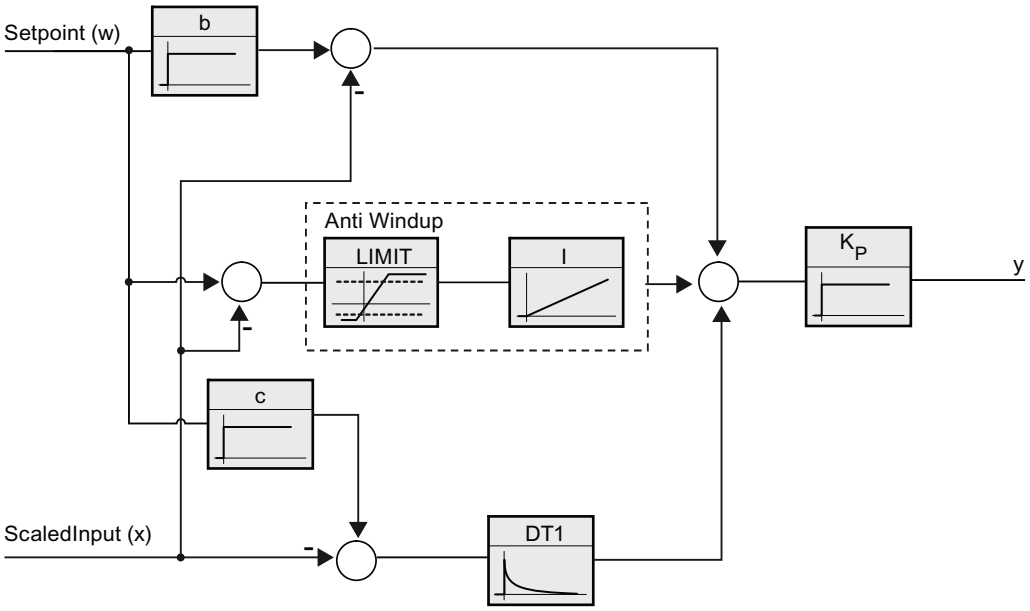


Figure 9-2 Operation of the PID_Compact controller as a PIDT1 controller with anti-windup

9.2.3 PID_3STEP instruction

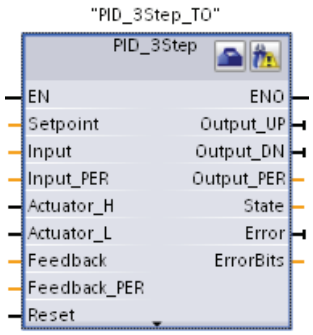
The PID controller uses the following formula to calculate the output value for the PID_3Step instruction.

$$\Delta y = K_p \cdot s \cdot \left[(b \cdot w - x) + \frac{1}{T_I \cdot s} (w - x) + \frac{T_D \cdot s}{a \cdot T_D \cdot s + 1} (c \cdot w - x) \right]$$

y	Output value	x	Process value
w	Setpoint value	s	Laplace operator
K_p	Proportional gain (P component)	a	Derivative delay coefficient (D component)
T_1	Integral action time (I component)	b	Proportional action weighting (P component)
T_D	Derivative action time (D component)	c	Derivative action weighting (D component)

Table 9- 15 PID_3Step instruction

LAD / FBD	SCL	Description

	`"PID_3Step_1"(`
	`SetpoInt:=_real_in_,`
	`Input:=_real_in_,`
	`ManualValue:=_real_in_,`
	`Feedback:=_real_in_,`
	`InputPer:=_word_in_,`
	`FeedbackPer:=_word_in_,`
	`ManualEnable:=_bool_in_,`
	`ManualUP:=_bool_in_,`
	`ManualDN:=_bool_in_,`
	`ActuatorH:=_bool_in_,`
	`ActuatorL:=_bool_in_,`
	`Reset:=_bool_in_,`
	`ScaledInput=>_real_out_,`
	`ScaledFeedback=>_real_out_,`
	`ErrorBits=>_dword_out_,`
	`OutputPer=>_word_out_,`
	`State=>_int_out_,`
	`OutputUP=>_bool_out_,`
	`OutputDN=>_bool_out_,`
	`SetpoIntLimitH=>_bool_out_,`
	`SetpoIntLimitL=>_bool_out_,`
	`InputWarningH=>_bool_out_,`
	`InputWarningL=>_bool_out_,`
	`Error=>_bool_out_);`
		PID_3Step configures a PID controller with self-tuning capabilities that has been optimized for motor-controlled valves and actuators. It provides two Boolean outputs.
		PID_3Step is a PIDT1controller with anti-windup and weighting of the P- and D-components.

1 STEP 7 automatically creates the technological object and instance DB when you insert the instruction. The instance DB contains the parameters of the technological object.

2 In the SCL example, "PID_3Step_1" is the name of the instance DB.

Table 9- 16 Data types for the parameters

Parameter and type		Data type	Description
Setpoint	IN	Real	Setpoint of the PID controller in automatic mode. Default value: 0.0
Input	IN	Real	Process value. Default value: 0.0 You must also set Config.InputPEROn = FALSE.
Input_PER	IN	Word	Analog process value (optional). Default value: W#16#0 You must also set Config.InputPEROn = TRUE.
ManualEnable	IN	Bool	Enables or disables the manual operation mode. Default value: FALSE • On the edge of the change from FALSE to TRUE, the PID controller switches to manual mode, State = 4, and Retain.Mode remains unchanged. • On the edge of the change from TRUE to FALSE, the PID controller switches to the last active operating mode and State = Retain.Mode.
ManualUP	IN	Bool	In manual mode, every rising edge opens the valve by 5% of the total actuating range, or for the duration of the minimum motor actuation time. ManualUP is evaluated only if you are using OutputPer **and** if position feedback is available. Default value: FALSE • If Output_PER is FALSE, the manual input turns Output_UP on for the time that corresponds to a movement of 5% of the device. • If Config.ActuatorEndStopOn is TRUE, then Output_UP does not come on if Actuator_H is TRUE.
ManualDN	IN	Bool	In manual mode, every rising edge closes the valve by 5% of the total actuating range, or for the duration of the minimum motor actuation time. ManualDN is evaluated only if you are using OutputPer **and** if position feedback is available. Default value: FALSE • If Output_PER is FALSE, the manual input turns Output_DN on for the time that corresponds to a movement of 5% of the device. • If Config.ActuatorEndStopOn is TRUE, then Output_DN does not turn on if Actuator_L is TRUE.
ManualValue	IN	Real	Process value for manual operation. Default value: 0.0 In manual mode, you specify the absolute position of the valve. ManualValue is evaluated only if you are using OutputPer, **or** if position feedback is available. Default value: 0.0
Feedback	IN	Real	Position feedback of the valve. Default value: 0.0 To use Feedback, then set Config.FeedbackPerOn = FALSE.
Feedback_PER	IN	Word	Analog feedback of the valve position. Default value: W#16#0 To use Feedback_PER, set Config.FeedbackPerOn = TRUE. Feedback_PER is scaled, using the following parameters: • Config.FeedbackScaling.LowerPointIn • Config.FeedbackScaling.UpperPointIn • Config.FeedbackScaling.LowerPointOut • Config.FeedbackScaling.UpperPointOut
Actuator_H	IN	Bool	If Actuator_H = TRUE, the valve is at the upper end stop and is no longer moved in this direction. Default value: FALSE

Parameter and type		Data type	Description
Actuator_L	IN	Bool	If Actuator_L = TRUE, the valve is at the lower end stop and is no longer moved in this direction. Default value: FALSE
Reset	IN	Bool	Restarts the PID controller. Default value: FALSE If Reset = TRUE: • "Inactive" operating mode • Input value = 0 • Interim values of the controller are reset. (PID parameters are retained.)
ScaledInput	OUT	Real	Scaled process value
ScaledFeedback	OUT	Real	Scaled valve position
Output_PER	OUT	Word	Analog output value. If Config.OutputPerOn = TRUE, then Output_PER is evaluated.
Output_UP	OUT	Bool	Digital output value for opening the valve. Default value: FALSE If Config.OutputPerOn = FALSE, then parameter Output_UP is evaluated.
Output_DN	OUT	Bool	Digital output value for closing the valve. Default value: FALSE If Config.OutputPerOn = FALSE, then parameter Output_DN is evaluated.
SetpointLimitH	OUT	Bool	Setpoint high limit. Default value: FALSE If SetpointLimitH = TRUE, the absolute upper limit of the setpoint is reached. In the CPU, the setpoint is limited to the configured absolute upper limit of the actual value.
SetpointLimitL	OUT	Bool	Setpoint low limit. Default value: FALSE If SetpointLimitL = TRUE, the absolute lower limit of the setpoint is reached. In the CPU the setpoint is limited to the configured absolute lower limit of the actual value.
InputWarningH	OUT	Bool	If InputWarningH = TRUE, the input value has reached or exceeded the upper warning limit. Default value: FALSE
InputWarningL	OUT	Bool	If InputWarningL = TRUE, the input value has reached or exceeded the lower warning limit. Default value: FALSE
State	OUT	Int	Current operating mode of the PID controller. Default value: 0 Use Retain.Mode to change the operating mode: • State = 0: Inactive • State = 1: Pretuning • State = 2: Manual fine tuning • State = 3: Automatic mode • State = 4: Manual mode • State = 5: Safety mode • State = 6: Output value measurement • State = 7: Safety mode monitoring with active trigger • State = 8: Inactive mode monitoring with active trigger
Error	OUT	Bool	If Error = TRUE, at least one error message is pending. Default value: FALSE
ErrorBits	OUT	DWord	Error message. Default value: DW#16#0000 (no error)

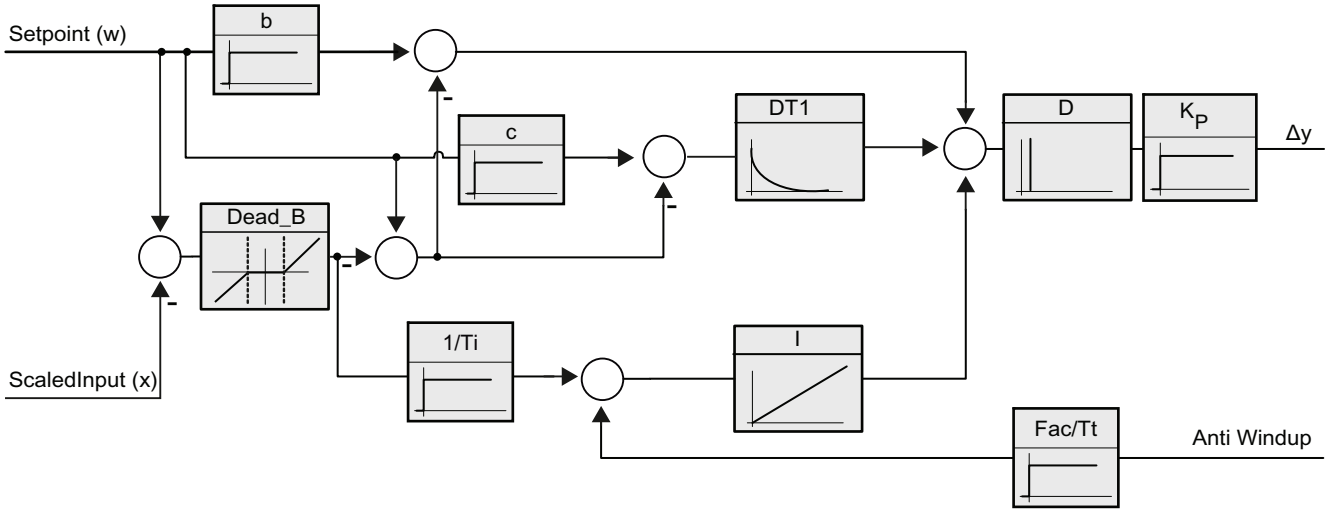


Figure 9-3 Operation of the PID_3Step controller as a PIDT1 controller with anti-windup

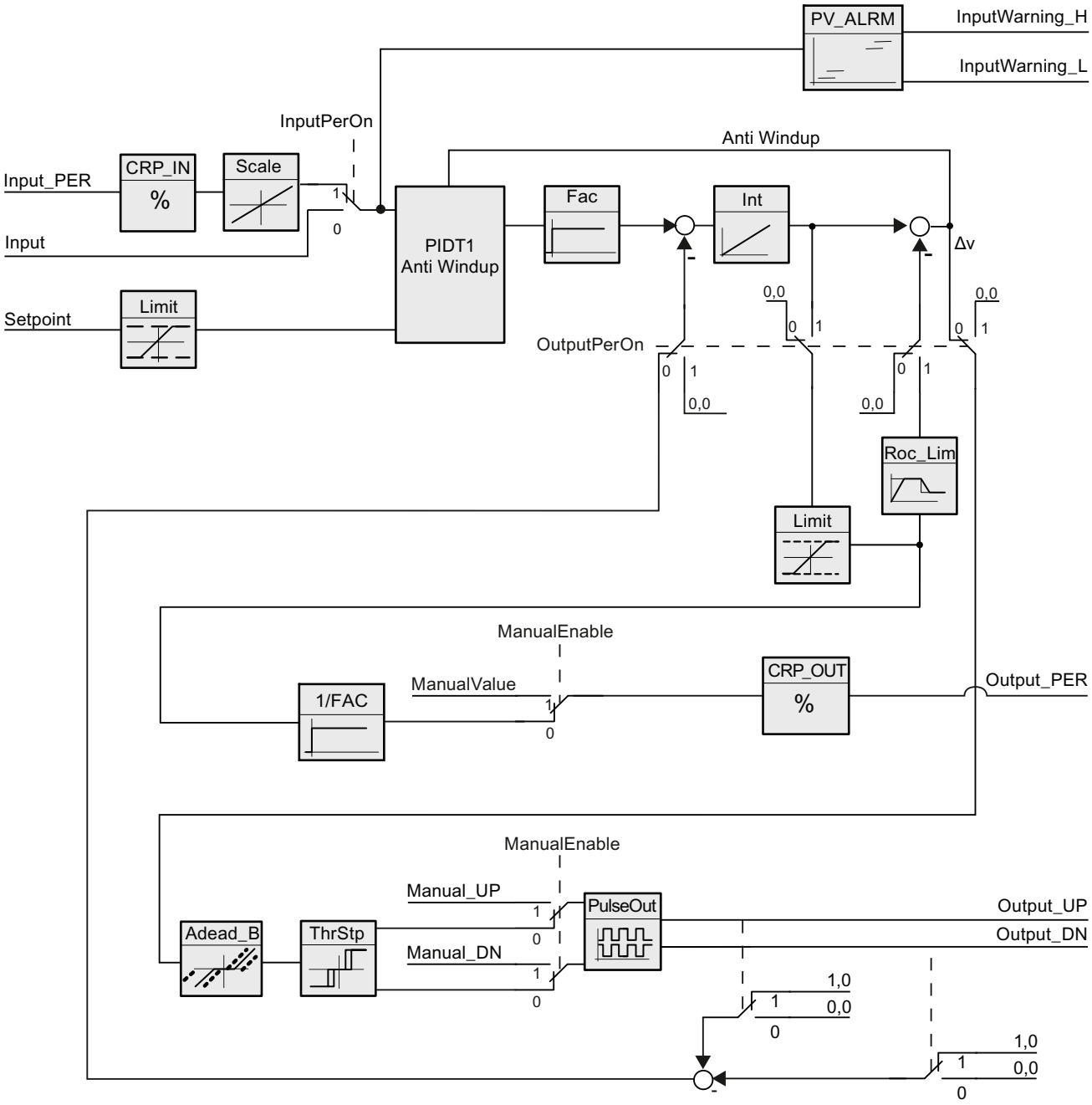


Figure 9-4 Operation of the PID_3Step controller without position feedback

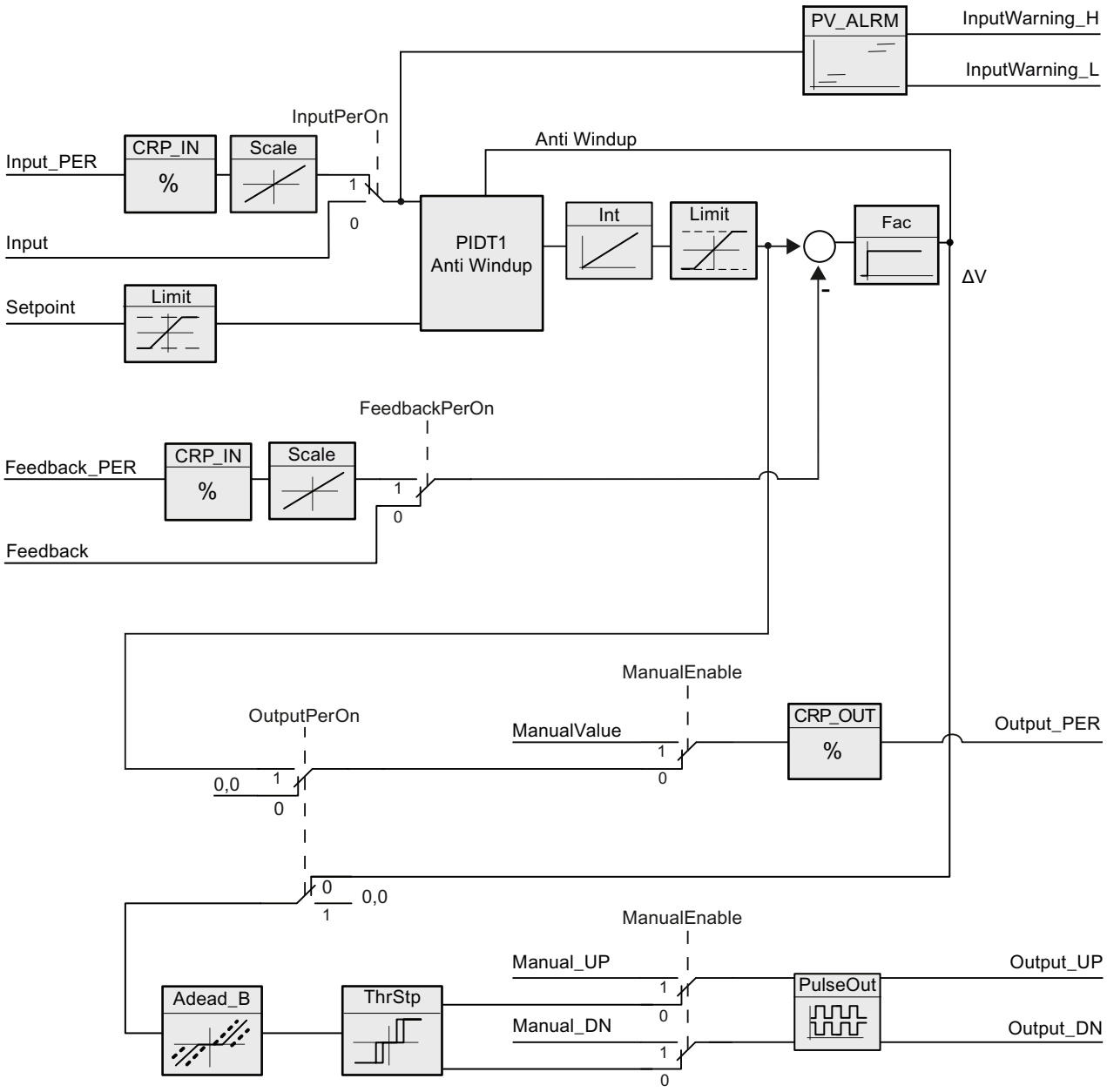


Figure 9-5 Operation of the PID_3Step controller the position feedback enabled

If several errors are pending, the values of the error codes are displayed by means of binary addition. The display of error code 0003, for example, indicates that the errors 0001 and 0002 are also pending.

Table 9- 17 ErrorBit parameters

ErrorBit (DW#16#...)	Description
0000	No error
0001	The Input parameter is outside the limits of the process value. • Input > Config.InputUpperLimit or • Input < Config.InputLowerLimit
0002	Invalid value for the "Input_PER" parameter. Determine whether there is an error at the analog input.
0004	Fine tuning: Oscillation of the process value (input) could not be maintained.
0008	Pretuning: The process value (input) is too close to the setpoint. Start the fine tuning.
0010	The setpoint must not be changed during pretuning at the operating point.
0020	Pretuning is set to automatic mode, which is not allowed during fine tuning.
0040	Pretuning: The setpoint is too close to the limits for the output value.
0080	Pretuning: Incorrect configuration of the limits for the output value.
0100	An error during fine tuning: resulted in invalid parameters.
0200	Invalid value for the Input parameter: • Value outside the number range (less than $-1e^{12}$ or greater than $1e^{12}$) • Value with invalid number format
0400	Invalid value for the Output parameter: • Value outside the number range (less than $-1e^{12}$ or greater than $1e^{12}$) • Value with invalid number format
800	Sampling time error: The PID_3STEP instruction is called in a program cycle OB (such as OB 1), or the settings were changed for the cyclic interrupt OB.
1000	Invalid value for the Setpoint parameter: • Value outside the number range (less than $-1e^{12}$, or greater than $1e^{12}$) • Value with invalid number format

9.2.4 Configuring the PID controller

The parameters of the technological object determine the operation of the PID controller. Use the icon to open the configuration editor.

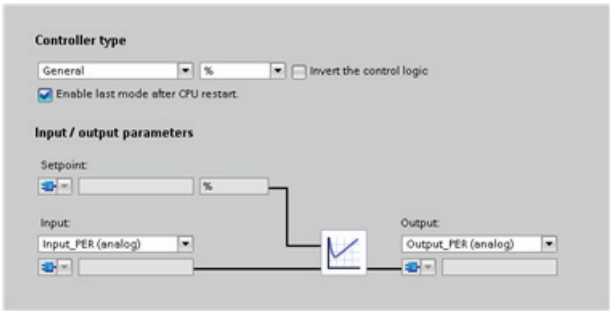


Figure 9-6 Configuration editor for PID_Compact (Basic settings)

Table 9- 18 Sample configuration settings for the PID_Compact instruction

Settings		Description
Basic	Controller type	Selects the engineering units.
	Invert the control logic	Allows selection of a reverse-acting PID loop. • If not selected, the PID loop is in direct-acting mode and the output of PID loop increases if input value < setpoint. • If selected, the output of the PID loop increases if the input value > setpoint.
	Enable last mode after CPU restart	Restarts the PID loop after it is reset or if an input limit has been exceeded and returned to the valid range.
	Input	Selects either the Input parameter or the Input_PER parameter (for analog) for the process value. Input_PER can come directly from an analog input module.
	Output	Selects either the Output parameter or the Output_PER parameter (for analog) for the output value. Output_PER can go directly to an analog output module.
Process value	Scales both the range and the limits for the process value. If the process value goes below the low limit or above the high limit, the PID loop goes to inactive mode and sets the output value to 0. To use Input_PER, you **must** scale the analog process value (input value).	

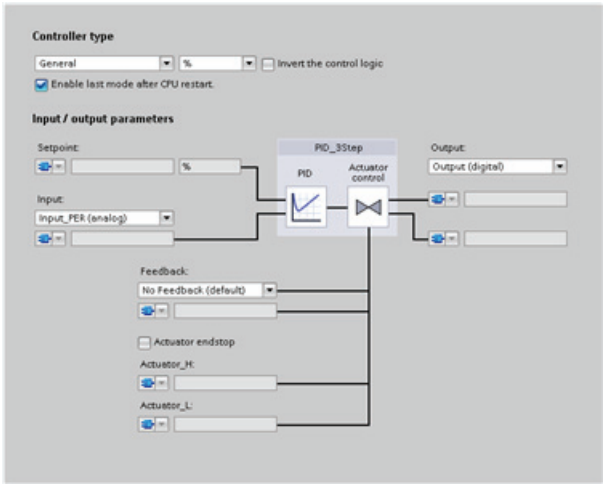


Figure 9-7 Configuration editor for PID_3Step (Basic settings)

Table 9- 19 Sample configuration settings for the PID_3Step instruction

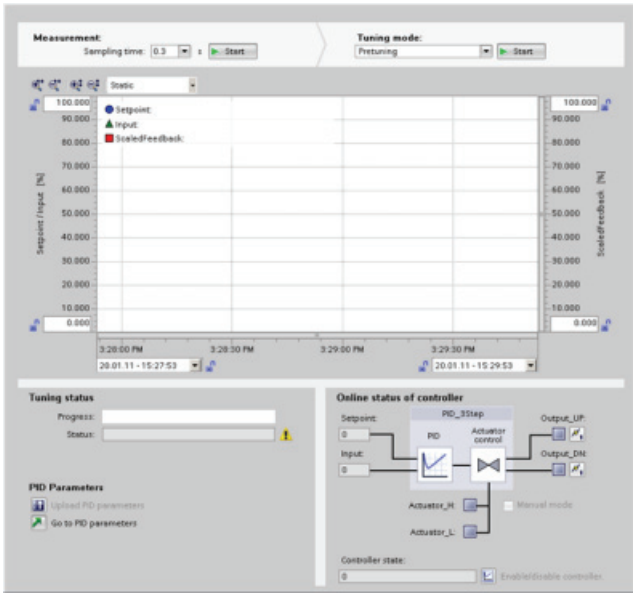
Settings		Description
Basic	Controller type	Selects the engineering units.
	Invert the control logic	Allows selection of a reverse-acting PID loop. • If not selected, the PID loop is in direct-acting mode, and the output of PID loop increases if the input value < setpoint). • If selected, the output of the PID loop increases if the input value > setpoint.
	Enable last mode after CPU restart	Restarts the PID loop after it is reset or if an input limit has been exceeded and returned to the valid range.
	Input	Selects either the Input parameter or the Input_PER parameter (for analog) for the process value. Input_PER can come directly from an analog input module.
	Output	Selects either to use the digital outputs (Output_UP and Output_DN) or to use the analog output (Output_PER) for the output value.
	Feedback	Selects the type of device status returned to the PID loop: • No feedback (default) • Feedback • Feedback_PER
Process value	Scales both the range and the limits for the process value. If the process value goes below the low limit or above the high limit, the PID loop goes to inactive mode and sets the output value to 0. To use Input_PER, you **must** scale the analog process value (input value).	
Actuator	Motor transition time	Sets the time from open to close for the valve. (Locate this value on the data sheet or the faceplate of the valve.)
	Minimum ON time	Sets the minimum movement time for the valve. (Locate this value on the data sheet or the faceplate of the valve.)
	Minimum OFF time	Sets the minimum pause time for the valve. (Locate this value on the data sheet or the faceplate of the valve.)
	Error behavior	Defines the behavior of the valve when an error is detected or when the PID loop is reset. If you select to use a substitute position, enter the "Safety position". For analog feedback or analog output, select a value between the upper or lower limit for the output. For digital outputs, you can choose only 0% (off) or 100% (on).
	Scale Position Feedback[1]	• "High stop" and "Lower limit stop" define the maximum positive position (full-open) and the maximum negative position (full-closed). "High stop" must be greater than "Lower limit stop". • "High limit process value" and "Low limit process value" define the upper and lower positions of the valve during tuning and automatic mode. • "FeedbackPER" ("Low" and "High") defines the analog feedback of the valve position. "FeedbackPER High" must be greater than "FeedbackPER Low".

[1] "Scale Position Feedback" is editable only if you enabled "Feedback" in the "Basic" settings.

9.2.5 Commissioning the PID controller

Use the commissioning editor to configure the PID controller for autotuning at startup and for autotuning during operation. To open the commissioning editor, click the icon on either the instruction or the project navigator.

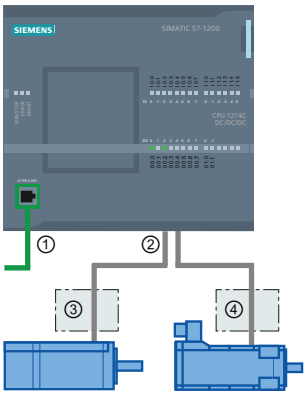
Table 9- 20 Sample configuration screen (PID_3Step)

	• Measurement: To display the setpoint, the process value (input value) and the output value in a real-time trend, enter the sample time and click the "Start" button. • Tuning mode: To tune the PID loop, select either "Pretuning" or "Fine tuning" (manual) and click the "Start" button. The PID controller runs through multiple phases to calculate system response and update times. The appropriate tuning parameters are calculated from these values. After the completion of the tuning process, you can store the new parameters by clicking the "Upload PID parameters" button in the "PID Parameters" section of the commissioning editor. If an error occurs during tuning, the output value of the PID goes to 0. The PID mode then is set to "inactive" mode. The status indicates the error.

9.3 Basic motion control

The CPU provides motion control functionality for the operation of stepper motors and servo motors with pulse interface. The motion control functionality takes over the control and monitoring of the drives.

- The "Axis" technology object configures the mechanical drive data, drive interface, dynamic parameters, and other drive properties.
- You configure the pulse and direction outputs of the CPU for controlling the drive.
- Your user program uses the motion control instructions to control the axis and to initiate motion tasks.
- Use the PROFINET interface to establish the online connection between the CPU and the programming device. In addition to the online functions of the CPU, additional commissioning and diagnostic functions are available for motion control.



① PROFINET

② Pulse and direction outputs

③ Power section for stepper motor

④ Power section for servo motor

The DC/DC/DC variants of the CPU S7-1200 have onboard outputs for direct control of drives. The relay variants of the CPU require the signal board with DC outputs for drive control.

A signal board (SB) expands the onboard I/O to include a few additional I/O points. An SB with 2 digital outputs can be used as pulse and direction outputs to control one motor. An SB with 4 digital outputs can be used as pulse and direction outputs to control two motors. Built-in relay outputs cannot be used as pulse outputs to control motors.

Note

Pulse-train outputs cannot be used by other instructions in the user program

When you configure the outputs of the CPU or signal board as pulse generators (for use with the PWM or basic motion control instructions), the corresponding output addresses (Q0.0 to Q0.3, Q4.0 to Q4.3) are removed from the Q memory and cannot be used for other purposes in your user program. If your user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.

Table 9- 21 Maximum number of controllable drives

Type of CPU		No SB installed	With an SB (2 x DC outputs)	With an SB (4 x DC outputs)
CPU 1211C	DC/DC/DC	2	2	2
	AC/DC/RLY	0	1	2
	DC/DC/RLY	0	1	2
CPU 1212C	DC/DC/DC	2	2	2
	AC/DC/RLY	0	1	2
	DC/DC/RLY	0	1	2
CPU 1214C	DC/DC/DC	2	2	2
	AC/DC/RLY	0	1	2
	DC/DC/RLY	0	1	2

Table 9- 22 Limit frequencies of pulse outputs

Pulse output	Frequency
Onboard	2 Hz ≤ f ≤ 100 KHz
Standard SB	2 Hz ≤ f ≤ 20 KHz
High-speed (200 KHz) SBs	MC V2 instructions: 2 Hz ≤ f ≤ 200 KHz MC V1 instructions: 2 Hz ≤ f ≤ 100 KHz [1]

[1] MC V1 instructions support a maximum frequency of 100 KHz.

NOTICE

The maximum pulse frequency of the pulse output generators is 100 KHz for the digital outputs of the CPU, 20 KHz for the digital outputs of the standard SB, and 200 KHz for the digital outputs of the high-speed SBs (or 100 KHz for MC V1 instructions). However, STEP 7 does **not** alert you when you configure an axis with a maximum speed or frequency that exceeds this hardware limitation. This could cause problems with your application, so always ensure that you do not exceed the maximum pulse frequency of the hardware.

1. Configure a pulse generator: Select the "Pulse generators (PTO/PWM)" properties for a CPU (in Device configuration) and enable a pulse generator. Two pulse generators are available for each S7-1200 CPU. In this same configuration area under "Pulse options", select Pulse generator used as: "PTO".
2. Add a Technological object:
   - In the Project tree, expand the node "Technological Objects" and select "Add new object".
   - Select the "Axis" icon (rename if required) and click "OK" to open the configuration editor for the axis object.
   - Display the "Select PTO for Axis Control" properties under the "Basic parameters" and select the configured PTO. Note the two Q outputs assigned for pulse and direction.
   - Configure the remaining Basic and Extended parameters.
3. Program your application: Insert the MC_Power instruction in a code block.
   - For the Axis input, select the axis technology object that you created and configured.
   - Setting the Enable input to TRUE allows the other motion instructions to function.
   - Setting the Enable input FALSE cancels the other motion instructions.

   Note

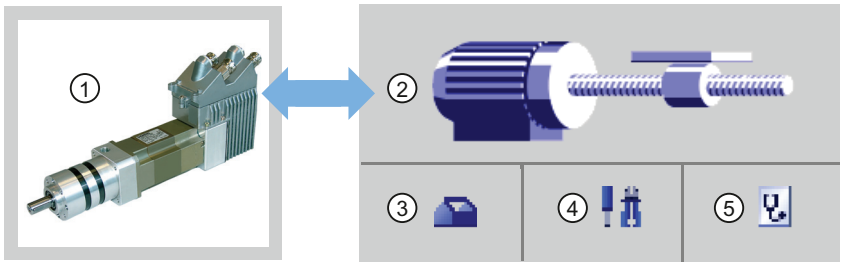
   Include only one MC_Power instruction per axis.
4. Insert the other motion instructions to produce the required motion.

Note

The CPU calculates motion tasks in "slices" or segments of 10 ms. As one slice is being executed, the next slice is waiting in the queue to be executed. If you interrupt the motion task on an axis (by executing another new motion task for that axis), the new motion task may not be executed for a maximum of 20 ms (the remainder of the current slice plus the queued slice).

9.3.1 Configuration of the axis

STEP 7 provides the configuration tools, the commissioning tools, and the diagnostic tools for the "Axis" technological object.



① Drive
② Technological object
③ Configuration
④ Commissioning
⑤ Diagnostics

Note

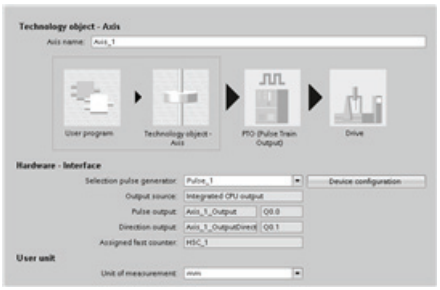
The PTO requires the internal functionality of a high-speed counter (HSC). This means the corresponding high-speed counter cannot be used elsewhere.

The assignment between PTO and HSC is fixed. When PTO1 is activated, it will be connected to HSC1. If PTO2 is activated, it will be connected to HSC2.

You cannot monitor the current value (for example, in ID 1000) when pulses are occurring.

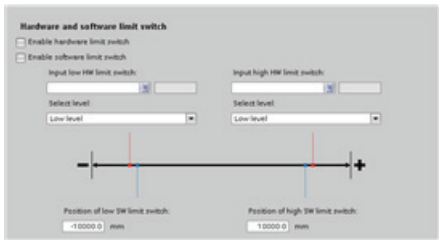
Table 9- 23 STEP 7 tools for motion control

Tool	Description
Configuration	Configures the following properties of the "Axis" technology object: • Selection of the PTO to be used and configuration of the drive interface • Properties of the mechanics and the transmission ratio of the drive (or machine or system) • Properties for position limits, dynamics, and homing Save the configuration in the data block of the technology object.
Commissioning	Tests the function of your axis without having to create a user program. When the tool is started, the control panel will be displayed. The following commands are available on the control panel: • Enable and disable axis • Move axis in jog mode • Position axis in absolute and relative terms • Home axis • Acknowledge errors The velocity and the acceleration / deceleration can be specified for the motion commands. The control panel also shows the current axis status.
Diagnostics	Monitors of the current status and error information for the axis and drive.



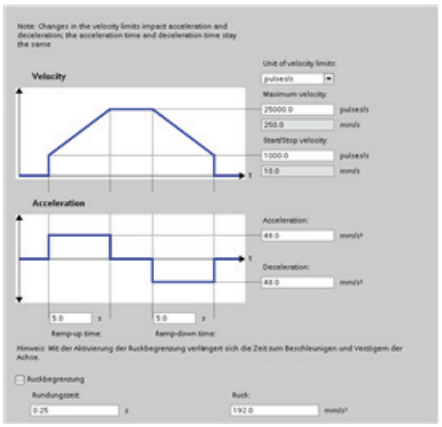
After you create the technological object for the axis, you configure the axis by defining the basic parameters, such as the PTO and the configuration of the drive interface. You also configure the other properties of the axis, such as position limits, dynamics, and homing.

NOTICE
If you change the dimension system at a later time, the values may not be converted correctly in all configuration windows of the technology object. In this case, check the configuration of all axis parameters. You may have to adapt the values of the input parameters of motion control instructions to the new dimension unit in the user program.

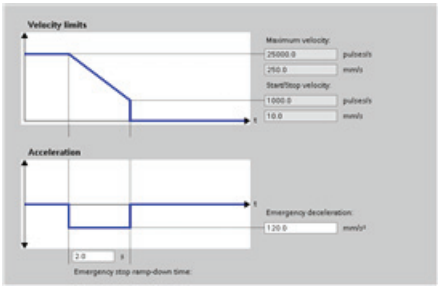


Configure the properties for the drive signals, drive mechanics, and position monitoring (hardware and software limit switches).

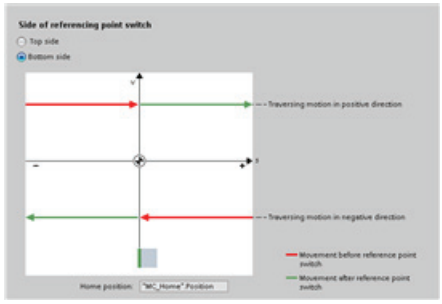
Do not deselect the options for a hardware limit or a reference point configuration unless the input point is no longer assigned as a hardware limit or a reference point.

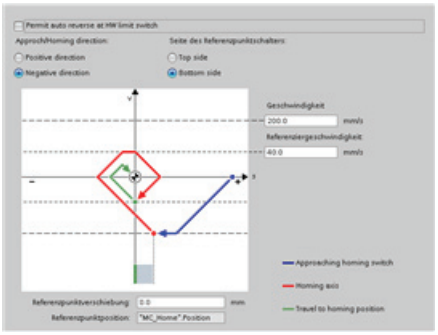


You configure the motion dynamics and the behavior of the emergency stop command.



You also configure the homing behavior (passive and active).





Use the "Commissioning" control panel to test the functionality independently from your user program.

Click the "Startup" icon to commission the axis.

The control panel shows the current status of the axis. Not only can you enable and disable the axis, but you can also test the positioning of the axis (both in absolute and relative terms) and can specify the velocity, acceleration and deceleration. You can also test the homing and jogging tasks. The control panel also allows you to acknowledge errors.

9.3.2 Motion control instructions

Note

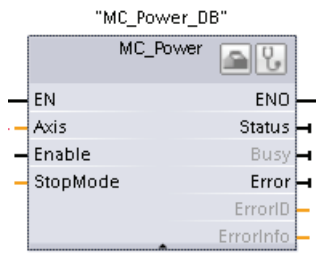
The CPU calculates motion tasks in "slices" or segments of 10 ms. As one slice is being executed, the next slice is waiting in the queue to be executed. If you interrupt the motion task on an axis (by executing another new motion task for that axis), the new motion task may not be executed for a maximum of 20 ms (the remainder of the current slice plus the queued slice).

9.3.2.1 MC_Power instruction

NOTICE

If the axis is switched off due to an error, it will be enabled again automatically after the error has been eliminated and acknowledged. This requires that the Enable input parameter has retained the value TRUE during this process.

Table 9- 24 MC_Power instruction

LAD / FBD	SCL	Description
	"MC_Power_DB"(Axis:=_multi_fb_in_, Enable:=_bool_in_, StopMode:=_int_in_, Status=>_bool_out_, Busy=>_bool_out_, Error=>_bool_out_, ErrorID=>_word_out_, ErrorInfo=>_word_out_);	The MC_Power motion control instruction enables or disables an axis. Before you can enable or disable the axis, ensure the following conditions: • The technology object has been configured correctly. • There is no pending enable-inhibiting error. The execution of MC_Power cannot be aborted by a motion control task. Disabling the axis (input parameter Enable = FALSE) aborts all motion control tasks for the associated technology object.

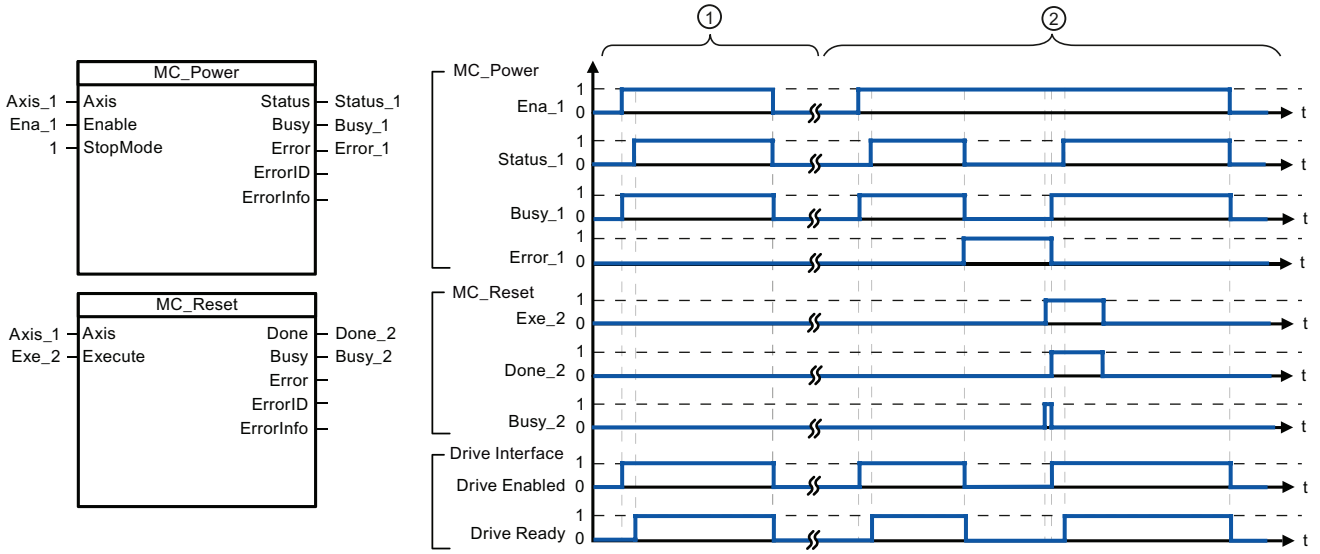
1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_Power_DB" is the name of the instance DB.

Table 9- 25 Parameters for the MC_Power instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
Enable	IN	Bool	• FALSE (default): All active tasks are aborted according to the parameterized "StopMode" and the axis is stopped. • TRUE: Motion Control attempts to enable the axis.

Parameter and type		Data type	Description
StopMode	IN	Int	• 0: Emergency stop - If a request to disable the axis is pending, the axis brakes at the configured emergency deceleration. The axis is disabled after reaching standstill. 1: Immediate stop - If a request to disable the axis is pending, this axis is disabled without deceleration. Pulse output is stopped immediately.
Status	OUT	Bool	Status of axis enable: • FALSE: The axis is disabled. – The axis does not execute motion control tasks and does not accept any new tasks (exception: MC_Reset task). – The axis is not homed. – Upon disabling, the status does not change to FALSE until the axis reaches a standstill. • TRUE: The axis is enabled. – The axis is ready to execute motion control tasks. – Upon axis enabling, the status does not change to TRUE until the signal "Drive ready" is pending. If the "Drive ready" drive interface was not configured in the axis configuration, the status changes to TRUE immediately.
Busy	OUT	Bool	False: MC_Power is not active. TRUE: MC Power is active.
Error	OUT	Bool	FALSE: No error TRUE: An error has occurred in motion control instruction "MC_Power" or in the associated technology object. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUT	Word	Error ID for parameter "Error""
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID"



① An axis is enabled and then disabled again. After the drive has signaled "Drive ready" back to the CPU, the successful enable can be read out via "Status_1".

② Following an axis enable, an error has occurred that caused the axis to be disabled. The error is eliminated and acknowledged with "MC_Reset". The axis is then enabled again.

To enable an axis with configured drive interface, follow these steps:

1. Check the requirements indicated above.
2. Initialize input parameter "StopMode" with the desired value. Set input parameter "Enable" to TRUE.

   The enable output for "Drive enabled" changes to TRUE to enable the power to the drive. The CPU waits for the "Drive ready" signal of the drive.

   When the "Drive ready" signal is available at the configured ready input of the CPU, the axis becomes enabled. Output parameter "Status" and technology object tag <Axis name>.StatusBits.Enable indicates the value TRUE.

To enable an axis without configured drive interface, follow these steps:

1. Check the requirements indicated above.
2. Initialize input parameter "StopMode" with the desired value. Set input parameter "Enable" to TRUE. The axis is enabled. Output parameter "Status" and technology object tag <Axis name>.StatusBits.Enable indicate the value TRUE.

To disable an axis, follow these steps:

1. Bring the axis to a standstill.

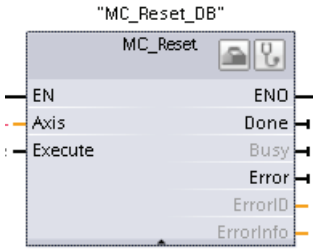
   You can identify when the axis is at a standstill in technology object tag <Axis name>.StatusBits.StandStill.

2. Set input parameter "Enable" to FALSE after standstill is reached.
3. If output parameters "Busy" and "Status" and technology object tag <Axis name>.StatusBits.Enable indicate the value FALSE, disabling of the axis is complete.

9.3.2.2 MC_Reset instruction

Table 9- 26 MC_Reset instruction

LAD / FBD	SCL	Description

	`"MC_Reset_DB"(`
	`Axis:=_multi_fb_in_,`
	`Execute:=_bool_in_,`
	`Done=>_bool_out_,`
	`Busy=>_bool_out_,`
	`Error=>_bool_out_,`
	`ErrorID=>_word_out_,`
	`ErrorInfo=>_word_out_);`
		Use the MC_Reset instruction to acknowledge "Operating error with axis stop" and "Configuration error". The errors that require acknowledgement can be found in the "List of ErrorIDs and ErrorInfos" under "Remedy".
		Before using the MC_Reset instruction, you must have eliminated the cause of a pending configuration error requiring acknowledgement (for example, by changing an invalid acceleration value in "Axis" technology object to a valid value).

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_Reset_DB" is the name of the instance DB.

The MC_Reset task cannot be aborted by any other motion control task. The new MC_Reset task does not abort any other active motion control tasks.

Table 9- 27 Parameters of the MC_Reset instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
Execute	IN	Bool	Start of the task with a positive edge
Done	OUT	Bool	TRUE = Error has been acknowledged.
Busy	OUT	Bool	TRUE = The task is being executed.
Error	OUT	Bool	TRUE = An error has occurred during execution of the task. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUTP	Word	Error ID for parameter "Error""
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID"

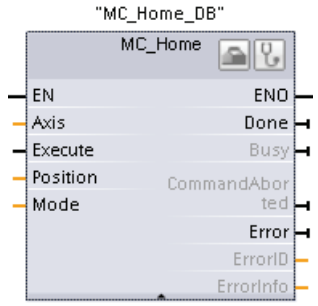
To acknowledge an error with MC_Reset, follow these steps:

1. Check the requirements indicated above.
2. Start the acknowledgement of the error with a rising edge at the Execute input parameter.
3. The error has been acknowledged when Done equals TRUE and the technology object tag <Axis name>.StatusBits.Error equals FALSE.

9.3.2.3 MC_Home instruction

Table 9- 28 MC_Home instruction

LAD / FBD	SCL	Description

	`"MC_Home_DB"(`
	`Axis:=_multi_fb_in_,`
	`Execute:=_bool_in_,`
	`Position:=_real_in_,`
	`Mode:=_int_in_,`
	`Done=>_bool_out_,`
	`Busy=>_bool_out_,`
	`CommandAborted=>_bool_out_,`
	`Error=>_bool_out_,`
	`ErrorID=>_word_out_,`
	`ErrorInfo=>_word_out_);`
		Use the MC_Home instruction to match the axis coordinates to the real, physical drive position. Homing is required for absolute positioning of the axis:
		In order to use the MC_Home instruction, the axis must first be enabled.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_Home_DB" is the name of the instance DB.

The following types of homing are available:

- Direct homing absolute (Mode = 0): The current axis position is set to the value of parameter "Position".
- Direct homing relative (Mode = 1): The current axis position is offset by the value of parameter "Position".
- Passive homing (Mode = 2): During passive homing, the MC_Home instruction does not carry out any homing motion. The traversing motion required for this step must be implemented by the user via other motion control instructions. When the reference point switch is detected, the axis is homed.
- Active homing (Mode = 3): The homing procedure is executed automatically.

Table 9- 29 Parameters for the MC_Home instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_PTO	Axis technology object
Execute	IN	Bool	Start of the task with a positive edge
Position	IN	Real	• Mode = 0, 2, and 3 (Absolute position of axis after completion of the homing operation) • Mode = 1 (Correction value for the current axis position) Limit values: $-1.0e^{12} \leq$ Position $\leq 1.0e^{12}$

Parameter and type		Data type	Description
Mode	IN	Int	Homing mode • 0: Direct homing absolute New axis position is the position value of parameter "Position". • 1: Direct homing relative New axis position is the current axis position + position value of parameter "Position". • 2: Passive homing Homing according to the axis configuration. Following homing, the value of parameter "Position" is set as the new axis position. • 3: Active homing Reference point approach in accordance with the axis configuration. Following homing, the value of parameter "Position" is set as the new axis position.
Done	OUT	Bool	TRUE = Task completed
Busy	OUT	Bool	TRUE = The task is being executed.
CommandAborted	OUT	Bool	TRUE = During execution the task was aborted by another task.
Error	OUT	Bool	TRUE = An error has occurred during execution of the task. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUT	Word	Error ID for parameter "Error""
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID"

Note

Axis homing is lost under the following conditions

- Disabling of axis by the MC_Power instruction
- Switchover between automatic control and manual control
- Upon start of active homing (After successful completion of the homing operation, axis homing is available again.)
- After power-cycling the CPU
- After CPU restart (RUN-to-STOP or STOP-to-RUN)

To home the axis, follow these steps:

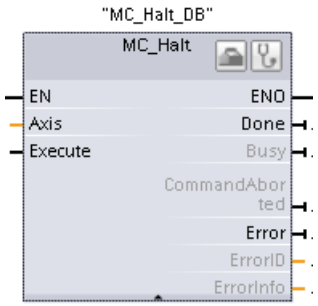
1. Check the requirements indicated above.
2. Initialize the necessary input parameters with values, and start the homing operation with a rising edge at input parameter "Execute".
3. If output parameter "Done" and technology object tag <Axis name>.StatusBits.HomingDone indicate the value TRUE, homing is complete.

Table 9- 30 Override response

Mode	Description	
0 or 1	The MC_Home task cannot be aborted by any other motion control task. The new MC_Home task does not abort any active motion control tasks. Position-related motion tasks are resumed after homing according to the new homing position (value at the Position input parameter).	
2	The MC_Home task can be aborted by the following motion control tasks: MC_Home task Mode = 2, 3: The new MC_Home task aborts the following active motion control task. MC_Home task Mode = 2: Position-related motion tasks are resumed after homing according to the new homing position (value at the Position input parameter).	
3	The MC_Home task can be aborted by the following motion control tasks: • MC_Home Mode = 3 • MC_Halt • MC_MoveAbsolute • MC_MoveRelative • MC_MoveVelocity • MC_MoveJog	The new MC_Home task aborts the following active motion control tasks: • MC_Home Mode = 2, 3 • MC_Halt • MC_MoveAbsolute • MC_MoveRelative • MC_MoveVelocity • MC_MoveJog

9.3.2.4 MC_Halt instruction

Table 9- 31 MC_Halt instruction

LAD / FBD	SCL	Description
	"MC_Halt_DB"(Axis:=_multi_fb_in_, Execute:=_bool_in_, Done=>_bool_out_, Busy=>_bool_out_, CommandAborted=>_bool_out_, Error=>_bool_out_, ErrorID=>_word_out_, ErrorInfo=>_word_out_);	Use the MC_Halt instruction to stop all motion and to brings the axis to a stand-still. The stand-still position is not defined. In order to use the MC_Halt instruction, the axis must first be enabled.

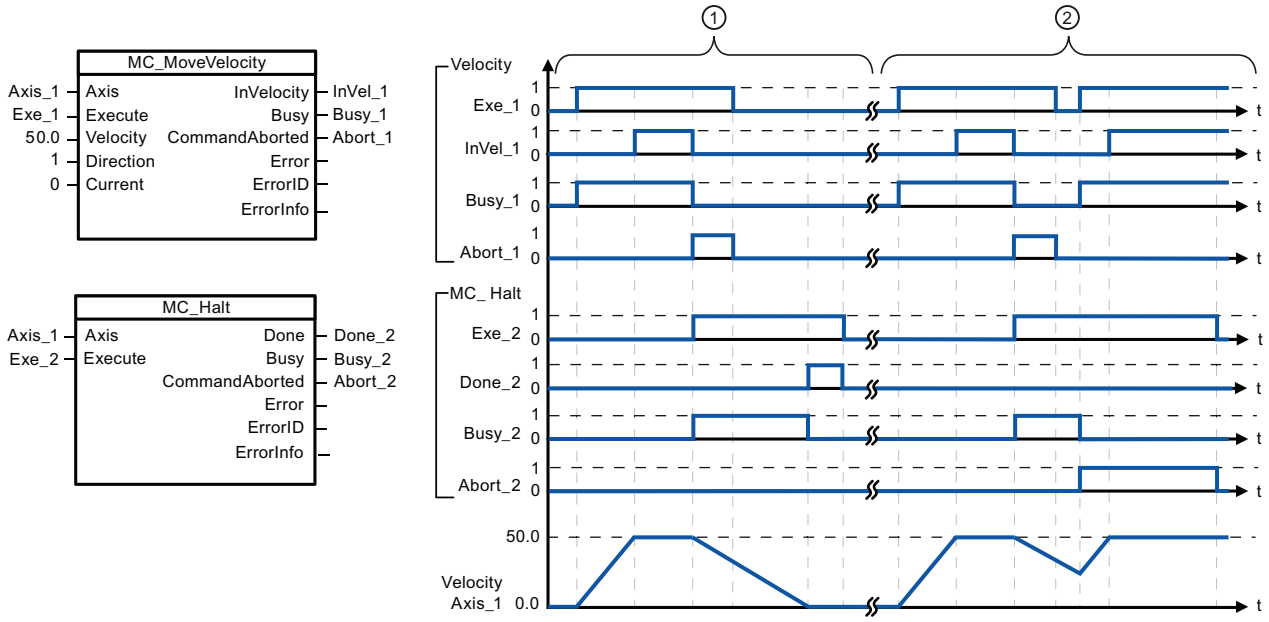
1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_Halt_DB" is the name of the instance DB.

Table 9- 32 Parameters for the MC_Halt instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
Execute	IN	Bool	Start of the task with a positive edge
Done	OUT	Bool	TRUE = Zero velocity reached
Busy	OUT	Bool	TRUE = The task is being executed.

Parameter and type		Data type	Description
CommandAborted	OUT	Bool	TRUE = During execution the task was aborted by another task.
Error	OUT	Bool	TRUE = An error has occurred during execution of the task. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUT	Word	Error ID for parameter "Error"
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID"



The following values were configured in the "Dynamics > General" configuration window: Acceleration = 10.0 and Deceleration = 5.0

① The axis is braked by an MC_Halt task until it comes to a standstill. The axis standstill is signaled via "Done_2".

② While an MC_Halt task is braking the axis, this task is aborted by another motion task. The abort is signaled via "Abort_2".

Override response

The MC_Halt task can be aborted by the following motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

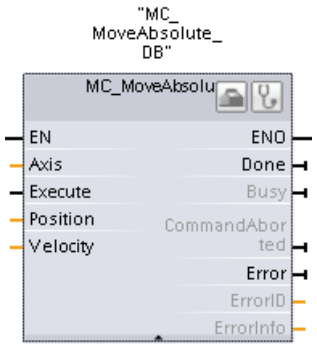
The new MC_Halt task aborts the following active motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

9.3.2.5 MC_MoveAbsolute instruction

Table 9- 33 MC_MoveAbsolute instruction

LAD / FBD	SCL	Description

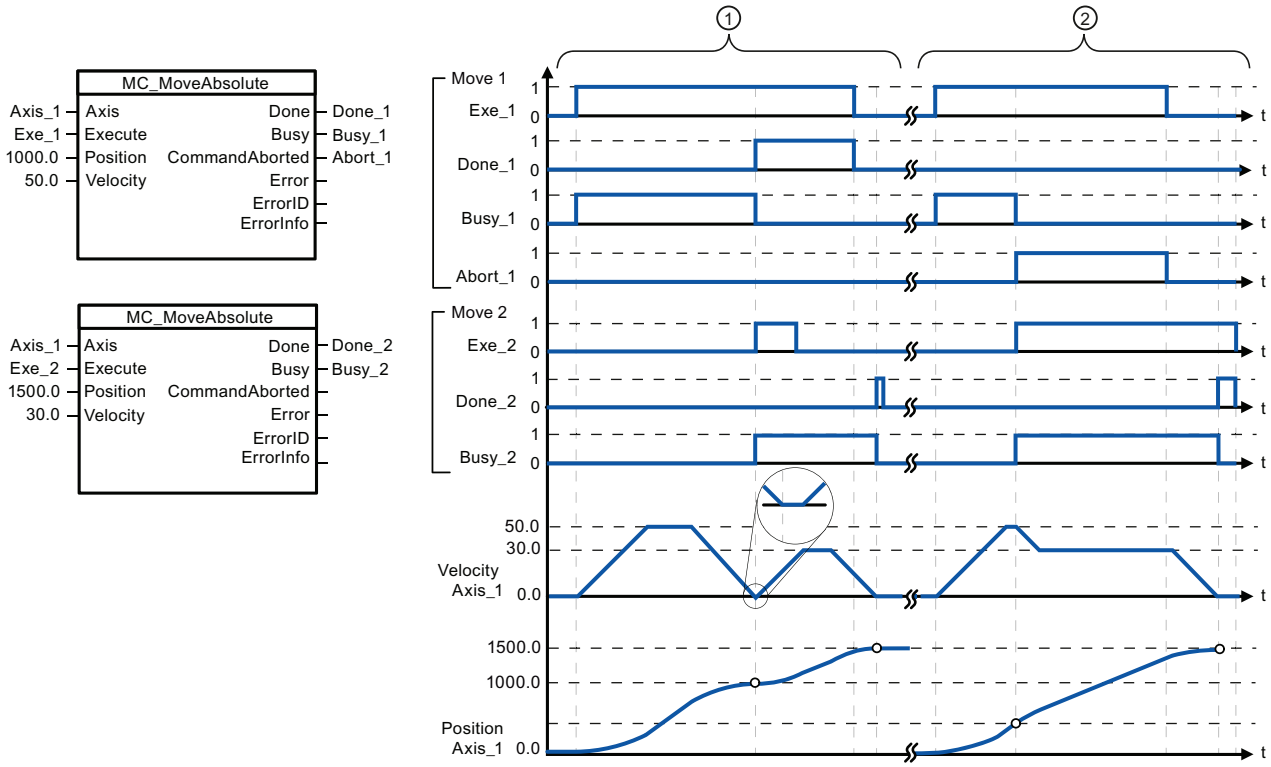
	`"MC_MoveAbsolute_DB"(`
	`Axis:=_multi_fb_in_,`
	`Execute:=_bool_in_,`
	`Position:=_real_in_,`
	`Velocity:=_real_in_,`
	`Done=>_bool_out_,`
	`Busy=>_bool_out_,`
	`CommandAborted=>_bool_out_,`
	`Error=>_bool_out_,`
	`ErrorID=>_word_out_,`
	`ErrorInfo=>_word_out_);`
		Use the MC_MoveAbsolute instruction to start a positioning motion of the axis to an absolute position.
		In order to use the MC_MoveAbsolute instruction, the axis must first be enabled and also must be homed.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_MoveAbsolute_DB" is the name of the instance DB.

Table 9- 34 Parameters for the MC_MoveAbsolute instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
Execute	IN	Bool	Start of the task with a positive edge (Default value: False)
Position	IN	Real	Absolute target position (Default value: 0.0) Limit values: $-1.0e^{12} \leq$ Position $\leq 1.0e^{12}$
Velocity	IN	Real	Velocity of axis (Default value: 10.0) This velocity is not always reached because of the configured acceleration and deceleration and the target position to be approached. Limit values: Start/stop velocity ≤ Velocity ≤ maximum velocity
Done	OUT	Bool	TRUE = Absolute target position reached
Busy	OUT	Bool	TRUE = The task is being executed.
CommandAborted	OUT	Bool	TRUE = During execution the task was aborted by another task.
Error	OUT	Bool	TRUE = An error has occurred during execution of the task. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUT	Word	Error ID for parameter "Error" (Default value: 0000)
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID" (Default value: 0000)



The following values were configured in the "Dynamics > General" configuration window: Acceleration = 10.0 and Deceleration = 10.0

① An axis is moved to absolute position 1000.0 with a MC_MoveAbsolute task. When the axis reaches the target position, this is signaled via "Done_1". When "Done_1" = TRUE, another MC_MoveAbsolute task, with target position 1500.0, is started. Because of the response times (e.g., cycle time of user program, etc.), the axis comes to a standstill briefly (see zoomed-in detail). When the axis reaches the new target position, this is signaled via "Done_2".

② An active MC_MoveAbsolute task is aborted by another MC_MoveAbsolute task. The abort is signaled via "Abort_1". The axis is then moved at the new velocity to the new target position 1500.0. When the new target position is reached, this is signaled via "Done_2".

Override response

The MC_MoveAbsolute task can be aborted by the following motion control tasks:

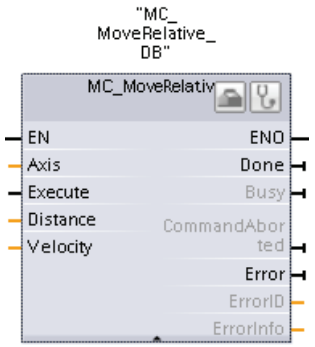
- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

The new MC_MoveAbsolute task aborts the following active motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

9.3.2.6 MC_MoveRelative instruction

Table 9- 35 MC_MoveRelative instruction

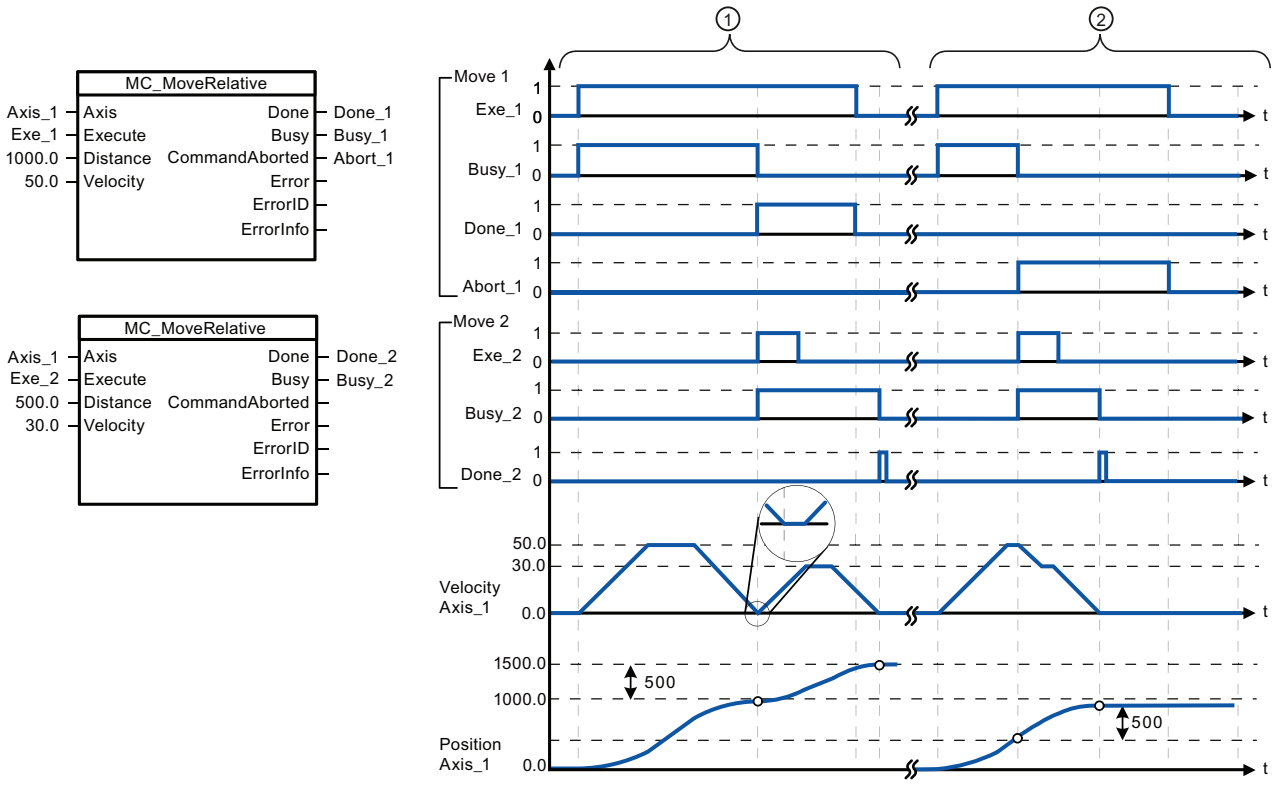
LAD / FBD	SCL	Description
	"MC_MoveRelative_DB"(Axis:=_multi_fb_in_, Execute:=_bool_in_, Distance:=_real_in_, Velocity:=_real_in_, Done=>_bool_out_, Busy=>_bool_out_, CommandAborted=>_bool_out_, Error=>_bool_out_, ErrorID=>_word_out_, ErrorInfo=>_word_out_);	Use the MC_MoveRelative instruction to start a positioning motion relative to the start position. In order to use the MC_MoveRelative instruction, the axis must first be enabled.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_MoveRelative_DB" is the name of the instance DB.

Table 9- 36 Parameters for the MC_MoveRelative instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
Execute	IN	Bool	Start of the task with a positive edge (Default value: False)
Distance	IN	Real	Travel distance for the positioning operation (Default value: 0.0) Limit values: $-1.0e^{12} \leq$ Distance $\leq 1.0e^{12}$
Velocity	IN	Real	Velocity of axis (Default value: 10.0) This velocity is not always reached on account of the configured acceleration and deceleration and the distance to be traveled. Limit values: Start/stop velocity ≤ Velocity ≤ maximum velocity
Done	OUT	Bool	TRUE = Target position reached
Busy	OUT	Bool	TRUE = The task is being executed.
CommandAborted	OUT	Bool	TRUE = During execution the task was aborted by another task.
Error	OUT	Bool	TRUE = An error has occurred during execution of the task. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUT	Word	Error ID for parameter "Error" (Default value: 0000)
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID" (Default value: 0000)



The following values were configured in the "Dynamics > General" configuration window: Acceleration = 10.0 and Deceleration = 10.0

① The axis is moved by an MC_MoveRelative task by the distance ("Distance") 1000.0. When the axis reaches the target position, this is signaled via "Done_1". When "Done_1" = TRUE, another MC_MoveRelative task, with travel distance 500.0, is started. Because of the response times (for example, cycle time of user program), the axis comes to a standstill briefly (see zoomed-in detail). When the axis reaches the new target position, this is signaled via "Done_2".

② An active MC_MoveRelative task is aborted by another MC_MoveRelative task. The abort is signaled via "Abort_1". The axis is then moved at the new velocity by the new distance ("Distance") 500.0. When the new target position is reached, this is signaled via "Done_2".

Override response

The MC_MoveRelative task can be aborted by the following motion control tasks:

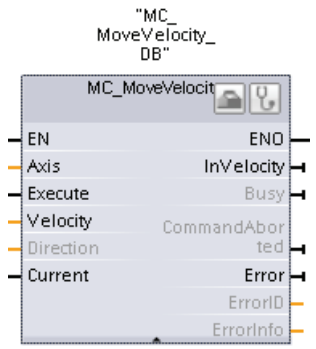
- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

The new MC_MoveRelative task aborts the following active motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

9.3.2.7 MC_MoveVelocity instruction

Table 9- 37 MC_MoveVelocity instruction

LAD / FBD	SCL	Description
	**"MC_MoveVelocity_DB"(** **Axis:=_multi_fb_in_,** **Execute:=_bool_in_,** **Velocity:=_real_in_,** **Direction:=_int_in_,** **Current:=_bool_in_,** **InVelocity=>_bool_out_,** **Busy=>_bool_out_,** **CommandAborted=>_bool_out_,** **Error=>_bool_out_,** **ErrorID=>_word_out_,** **ErrorInfo=>_word_out_);**	Use the MC_MoveVelocity instruction to move the axis constantly at the specified velocity. In order to use the MC_MoveVelocity instruction, the axis must first be enabled.

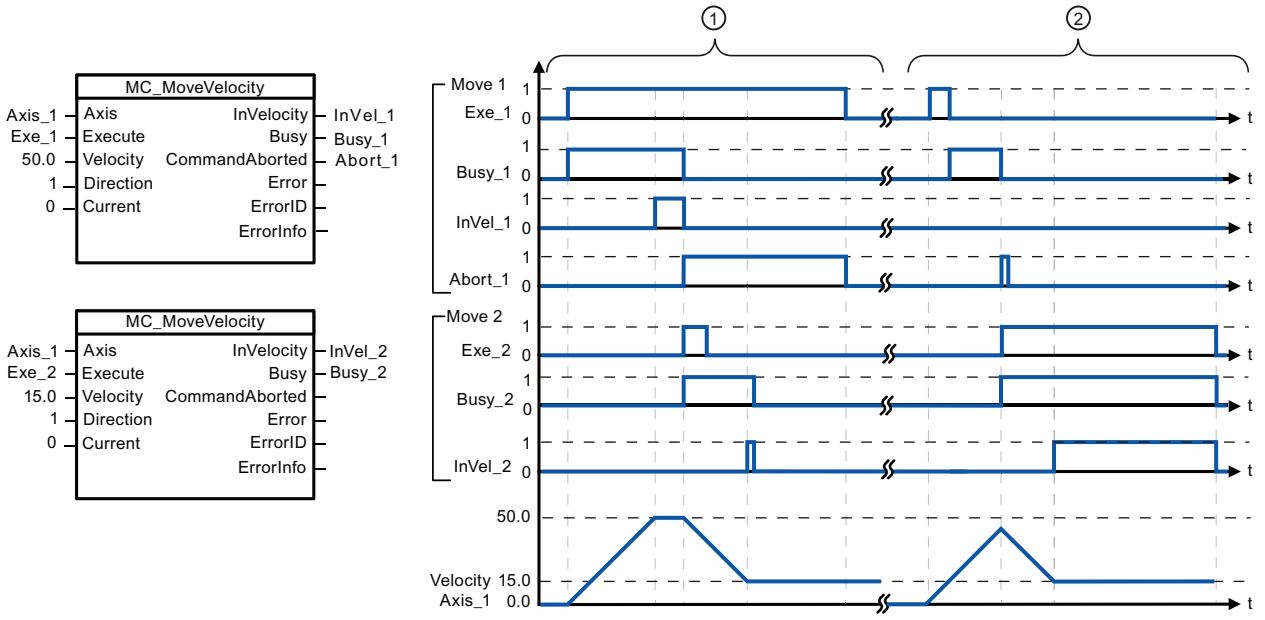
1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_MoveVelocity_DB" is the name of the instance DB.

Table 9- 38 Parameters for the MC_MoveVelocity instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
Execute	IN	Bool	Start of the task with a positive edge (Default value: False)
Velocity	IN	Real	Velocity specification for axis motion (Default value: 10.0) Limit values: Start/stop velocity ≤ |Velocity| ≤ maximum velocity (Velocity = 0.0 is allowed)
Direction	IN	Int	Direction specification: • 0: Direction of rotation corresponds to the sign of the value in parameter "Velocity" (Default value) • 1: Positive direction of rotation (The sign of the value in parameter "Velocity" is ignored.) • 2: Negative direction of rotation (The sign of the value in parameter "Velocity" is ignored.)
Current	IN	Bool	Maintain current velocity: • FALSE: "Maintain current velocity" is deactivated. The values of parameters "Velocity" and "Direction" are used. (Default value) • TRUE: "Maintain current velocity" is activated. The values in parameters "Velocity" and "Direction" are not taken into account. When the axis resumes motion at the current velocity, the "InVelocity" parameter returns the value TRUE.

Parameter and type		Data type	Description
InVelocity	OUT	Bool	TRUE: • If "Current" = FALSE: The velocity specified in parameter "Velocity" was reached. • If "Current" = TRUE: The axis travels at the current velocity at the start time.
Busy	OUT	Bool	TRUE = The task is being executed.
CommandAborted	OUT	Bool	TRUE = During execution the task was aborted by another task.
Error	OUT	Bool	TRUE = An error has occurred during execution of the task. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUT	Word	Error ID for parameter "Error" (Default value: 0000)
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID" (Default value: 0000)



The following values were configured in the "Dynamics > General" configuration window: Acceleration = 10.0 and Deceleration = 10.0

① An active MC_MoveVelocity task signals via "InVel_1" that its target velocity has been reached. It is then aborted by another MC_MoveVelocity task. The abort is signaled via "Abort_1". When the new target velocity 15.0 is reached, this is signaled via "InVel_2". The axis then continues moving at the new constant velocity.

② An active MC_MoveVelocity task is aborted by another MC_MoveVelocity task prior to reaching its target velocity. The abort is signaled via "Abort_1". When the new target velocity 15.0 is reached, this is signaled via "InVel_2". The axis then continues moving at the new constant velocity.

Override response

The MC_MoveVelocity task can be aborted by the following motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

The new MC_MoveVelocity task aborts the following active motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

Note

Behavior with zero set velocity (Velocity = 0.0)

An MC_MoveVelocity task with "Velocity" = 0.0 (such as an MC_Halt task) aborts active motion tasks and stops the axis with the configured deceleration. When the axis comes to a standstill, output parameter "InVelocity" indicates TRUE for at least one program cycle.

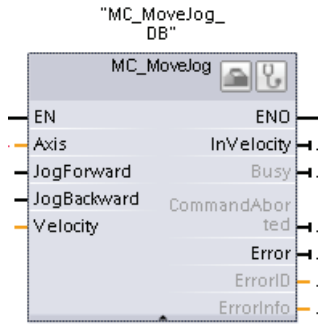
"Busy" indicates the value TRUE during the deceleration operation and changes to FALSE together with "InVelocity". If parameter "Execute" = TRUE is set, "InVelocity" and "Busy" are latched.

When the MC_MoveVelocity task is started, status bit "SpeedCommand" is set in the technology object. Status bit "ConstantVelocity" is set upon axis standstill. Both bits are adapted to the new situation when a new motion task is started.

9.3.2.8 MC_MoveJog instruction

Table 9- 39 MC_MoveJog instruction

LAD / FBD	SCL	Description

	`"MC_MoveJog_DB"(`
	`Axis:=_multi_fb_in_,`
	`JogForward:=_bool_in_,`
	`JogBackward:=_bool_in_,`
	`Velocity:=_real_in_,`
	`InVelocity=>_bool_out_,`
	`Busy=>_bool_out_,`
	`CommandAborted=>_bool_out_,`
	`Error=>_bool_out_,`
	`ErrorID=>_word_out_,`
	`ErrorInfo=>_word_out_);`
		Use the MC_MoveJog instruction to move the axis constantly at the specified velocity in jog mode. This instruction is typically used for testing and commissioning purposes.
		In order to use the MC_MoveJog instruction, the axis must first be enabled.

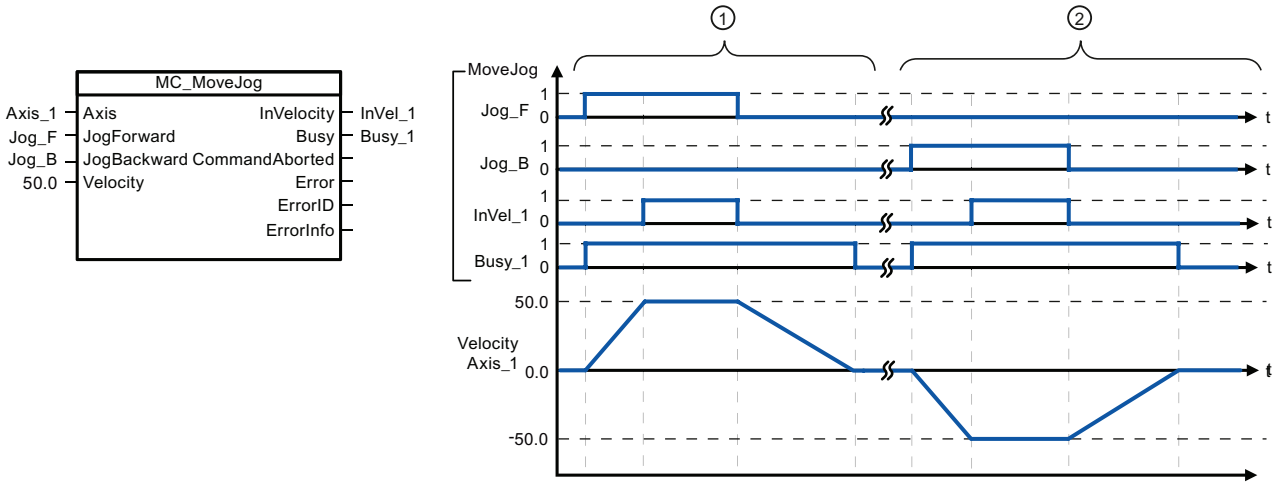
1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_MoveJog_DB" is the name of the instance DB.

Table 9- 40 Parameters for the MC_MoveJog instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
JogForward[1]	IN	Bool	As long as the parameter is TRUE, the axis moves in the positive direction at the velocity specified in parameter "Velocity". The sign of the value in parameter "Velocity" is ignored. (Default value: False)
JogBackward[1]	IN	Bool	As long as the parameter is TRUE, the axis moves in the negative direction at the velocity specified in parameter "Velocity". The sign of the value in parameter "Velocity" is ignored. (Default value: False)
Velocity	IN	Real	Preset velocity for jog mode (Default value: 10.0) Limit values: Start/stop velocity ≤ |Velocity| ≤ maximum velocity
InVelocity	OUT	Bool	TRUE = The velocity specified in parameter "Velocity" was reached.
Busy	OUT	Bool	TRUE = The task is being executed.
CommandAborted	OUT	Bool	TRUE = During execution the task was aborted by another task.
Error	OUT	Bool	TRUE = An error has occurred during execution of the task. The cause of the error can be found in parameters "ErrorID" and "ErrorInfo".
ErrorID	OUT	Word	Error ID for parameter "Error" (Default value: 0000)
ErrorInfo	OUT	Word	Error info ID for parameter "ErrorID" (Default value: 0000)

[1] If both the JogForward and JogBackward parameters are simultaneously TRUE, the axis stops with the configured deceleration. An error is indicated in parameters "Error", "ErrorID", and "ErrorInfo".



The following values were configured in the "Dynamics > General" configuration window: Acceleration = 10.0 and Deceleration = 5.0

① The axis is moved in the positive direction in jog mode via "Jog_F". When the target velocity 50.0 is reached, this is signaled via "InVelo_1". The axis brakes to a standstill again after Jog_F is reset.

② The axis is moved in the negative direction in jog mode via "Jog_B". When the target velocity 50.0 is reached, this is signaled via "InVelo_1". The axis brakes to a standstill again after Jog_B is reset.

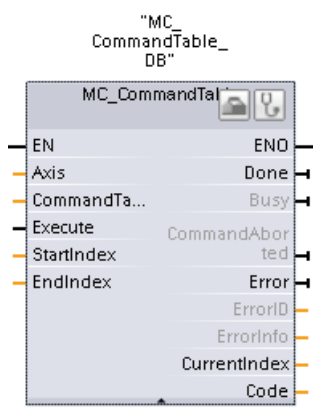
Override response

The MC_MoveJog task can be aborted by the following motion control tasks:	The new MC_MoveJog task aborts the following active motion control tasks:
• MC_Home Mode = 3 • MC_Halt • MC_MoveAbsolute • MC_MoveRelative • MC_MoveVelocity • MC_MoveJog	• MC_Home Mode = 3 • MC_Halt • MC_MoveAbsolute • MC_MoveRelative • MC_MoveVelocity • MC_MoveJog

9.3.2.9 MC_CommandTable instruction

Table 9- 41 MC_CommandTable instruction

LAD / FBD	SCL	Description

	`"MC_CommandTable_DB"(`
	`Axis:=_multi_fb_in_,`
	`CommandTable:=_multi_fb_in_,`
	`Execute:=_bool_in_,`
	`StartIndex:=_uint_in_,`
	`EndIndex:=_uint_in_,`
	`Done=>_bool_out_,`
	`Busy=>_bool_out_,`
	`CommandAborted=>_bool_out_,`
	`Error=>_bool_out_,`
	`ErrorID=>_word_out_,`
	`ErrorInfo=>_word_out_,`
	`CurrentIndex=>_uint_out_,`
	`Code=>_word_out_);`
		Executes a series of individual motions for a motor control axis that can combine into a movement sequence.
		Individual motions are configured in a technology object command table for pulse train output (TO_CommandTable_PTO).

1 STEP 7 automatically creates the DB when you insert the instruction.

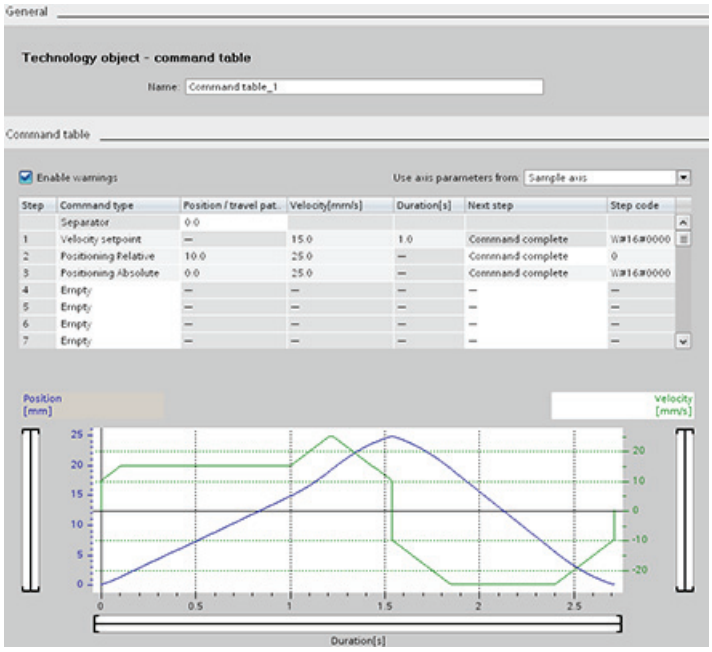
2 In the SCL example, "MC_CommandTable_DB" is the name of the instance DB.

Table 9- 42 Parameters for the MC_CommandTable instruction

Parameter and type		Data type	Initial value	Description
Axis	IN	TO_Axis_1	-	Axis technology object
Table	IN	TO_CommandTable_1	-	Command table technology object
Execute	IN	Booll	FALSE	Start job with rising edge
StartIndex	IN	Int	1	Start command table processing with this step Limits: 1 ≤ **StartIndex** ≤ EndIndex
EndIndex	IN	Int	32	End command table processing with this step Limits: StartIndex ≤ **EndIndex** ≤ 32
Done	OUT	Bool	FALSE	MC_CommandTable processing completed successfully

Parameter and type		Data type	Initial value	Description
Busy	OUT	Bool	FALSE	Operation in progress
CommandAborted	OUT	Bool	FALSE	The task was aborted during processing by another task.
Error	OUT	Bool	FALSE	An error ocurred during processing. The cause is indicated by the parameters ErrorID and ErrorInfo.
ErrorID	OUT	Word	16#0000	Error identifier
ErrorInfo	OUT	Word	16#0000	Error information
Step	OUT	Int	0	Step currently in process
Code	OUT	Word	16#0000	User defined identifier of the step currently in process

You can create the desired movement sequence in the "Command Table" configuration window and check the result against the graphic view in the trend diagram.



You can select the command types that are to be used for processing the command table. Up to 32 jobs can be entered. The commands are processed in sequence.

Table 9- 43 MC_CommandTable command types

Command type	Description
Empty	The empty serves as a placeholder for any commands to be added. The empty entry is ignored when the command table is processed
Halt	Pause axis. Note: The command only takes place after a "Velocity setpoint" command.
Positioning Relative	Positions the axis based upon distance. The command moves the axis by the given distance and velocity.
Positioning Absolute	Positions the axis based upon location. The command moves the axis by the given location and velocity.

Command type	Description
Velocity setpoint	Moves the axis at the given velocity.
Wait	Waits until the given period is over. "Wait" does not stop an active traversing motion.
Separator	Adds a "Separator" line above the selected line. The separator line acts as a range limit for the graphic display of the trend view.

Prerequisites for MC_CommandTable execution:

- The technology object TO_Axis_PTO V2.0 must be correctly configured.
- The technology object TO_CommandTable_PTO must be correctly configured.
- The axis must be released.

Override response

The MC_CommandTable task can be aborted by the following motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog

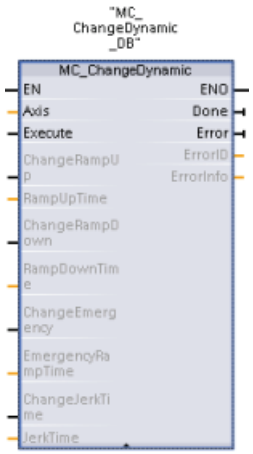
The new MC_CommandTable task aborts the following active motion control tasks:

- MC_Home Mode = 3
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative
- MC_MoveVelocity
- MC_MoveJog
- MC_CommandTable
- The current motion control job with the launch of the first "Positioning Relative", "Positioning Absolute", "Velocity setpoint" or "Halt" command

9.3.2.10 MC_ChangeDynamic

Table 9- 44 MC_ChangeDynamic instruction

LAD / FBD	SCL	Description

	`"MC_ChangeDynamic_DB"(`
	`Execute:=_bool_in_,`
	`ChangeRampUp:=_bool_in_,`
	`RampUpTime:=_real_in_,`
	`ChangeRampDown:=_bool_in_,`
	`RampDownTime:=_real_in_,`
	`ChangeEmergency:=_bool_in_,`
	`EmergencyRampTime:=_real_in_,`
	`ChangeJerkTime:=_bool_in_,`
	`JerkTime:=_real_in_,`
	`Done=>_bool_out_,`
	`Error=>_bool_out_,`
	`ErrorID=>_word_out_,`
	`ErrorInfo=>_word_out_);`
		Changes the dynamic settings of a motion control axis.:
		• Value for acceleration change
		• Value for delay change
		• Value for emergency stop-delay change
		• Jerk limitation activate / deactivate
		• Value for jerk limitation change

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "MC_ChangeDynamic_DB" is the name of the instance DB.

Table 9- 45 Parameters for the MC_ChangeDynamic instruction

Parameter and type		Data type	Description
Axis	IN	TO_Axis_1	Axis technology object
Execute	IN	Bool	Start execution on rising edge. Inital value: FALSE
ChangeRampUp	IN	Bool	TRUE = Change Ramp up time to the value specified by parameter RampUp Time. Inital value: FALSE
RampUpTime	IN	Real	Time (in seconds) without jerk limitation to accelerate from standstill to the configured maximum speed. Inital value: 5.00 You can see the status affected variable in the value of <Axisname>. Config.DynamicDefaults.Acceleration.
ChangeRampDown	IN	Bool	TRUE Change the release time according to the input parameter RampDownTime. Inital value: FALSE
RampDownTime	IN	Real	Time (in seconds) without jerk limitation to slow to a standstill from the configured maximum speed. Inital value: 5.00 You can see the status of the affected variable in the value of <Axisname>. Config.DynamicDefaults.Deceleration .
ChangeEmergency	IN	Bool	TRUE Change emergency stop time corresponding to the input parameter EmergencyRampTime. Inital value: FALSE
EmergencyRampTime	IN	Real	Time (in seconds) without jerk limitation for the emergency stop mode delay to go from the configured maximum speed to a standstill. Inital value: 2.00 Status of the affected variable stored in: <Axisname>. Config.DynamicDefaults.EmergencyDeceleration

Parameter and type		Data type	Description
ChangeJerkEnable	IN	Bool	TRUE = Change the jerk limitation according to the input parameter JerkEnable. Inital value: FALSE
JerkEnable	IN	Bool	TRUE = Enable the jerk limitation. Inital value: FALSE Status of the affected variable stored in: <Axisname>. Config.DynamicDefaults.JerkActive.
ChangeRoundingOff	IN	Bool	TRUE Change ramp in according to the input parameter RoundingOffTime. Inital value: FALSE
RoundingOffTime	IN	Real	Ramp (in seconds) that is applied to the acceleration and deceleration of the axis. Inital value: 0.20 Status of the affected variable stored in: <Axisname>. Config.DynamicDefaults.Jerk .
Done	OUT	Bool	TRUE = The modified values are written to the technology DB. Inital value: FALSE
Error	OUT	Bool	TRUE = An error has occurred during processing. The cause of the error can be found in the ErrorID and ErrorInfo parameters. Inital value: FALSE
ErrorID	OUT	Word	Error identifier. Inital value: 16#0000
ErrorInfo	IN	Word	Error information. Inital value: 16#0000

Prerequisites for MC_ ChangeDynamic execution:

- The technology object TO_Axis_PTO V2.0 must be correctly configured.
- The axis must be released.

Override response

Override response

The MC_ChangeDynamic task can be stopped by any other motion control task.	A new MC_ChangeDynamic task cancels any current motion control tasks.

Note

The input parameters "RampUpTime", "RampDownTime", "EmergencyRampTime" and "RoundingOffTime" can be specified with values that makes the resultant axis parameters "acceleration", "delay", "emergency stop-delay" and "jerk" outside the permissible limits.

Make sure you keep the MC_ChangeDynamic parameters within the limits of the dynamic configuration settings for the axis technological object.

9.3.3 Operation of motion control for S7-1200

9.3.3.1 CPU outputs used for motion control

The CPU provides one pulse output and one direction output for controlling a stepper motor drive or a servo motor drive with pulse interface. The pulse output provides the drive with the pulses required for motor motion. The direction output controls the travel direction of the drive.

Pulse and direction outputs are permanently assigned to one another. Onboard CPU outputs and outputs of a signal board can be used as pulse and direction outputs. You select between onboard CPU outputs and outputs of the signal board during device configuration under Pulse generators (PTO/PWM) on the "Properties" tab.

Table 9- 46 Address assignments of the pulse and direction outputs

CPU type (CPU 1211C, CPU 212C, and CPU 1214C)		Outputs PTO1 [1, 2]		Outputs PTO2 [1, 2]	
		Pulse	Direction	Pulse	Direction
DC/DC/DC	Without signal board	Qx.0	Qx.1	Qx.2	Qx.3
	Signal boards DI2/DO2 [3]	Qx.0	Qx.1	Qx.2	Qx.3
		Qy.0	Qy.1		
	Signal boards DO4 [4]	Qx.0	Qx.1	Qx.2	Qx.3
		Qy.0	Qy.1	Qy.2	Qy.3
AC/DC/RLY	Without signal board	-	-	-	-
	Signal boards DI2/DO2	Qy.0	Qy.1	-	-
	Signal boards DO4	Qy.0	Qy.1	Qy.2	Qy.3
DC/DC/RLY	Without signal board	-	-	-	-
	Signal boards DI2/DO2	Qy.0	Qy.1	-	-
	Signal boards DO4	Qy.0	Qy.1	Qy.2	Qy.3

1 x = Initial byte address of onboard CPU outputs (default value = 0)

2 y = Initial byte address of signal board outputs (default value = 4)

3 If a DC/DC/DC CPU variant is used together with a DI2/DO2 signal board, the signals of the PTO1 output can use either the onboard CPU outputs (Qx.0 and Qx.1) or the outputs of the signal board (Qy.0 and Qy.1).

4 If a DC/DC/DC CPU variant is used together with a DO4 signal board, the signals of the PTO outputs can use either the onboard CPU outputs (Qx.0 and Qx.1 for PTO1 and Qx.2 and Qx.3 for PTO2) or the outputs of the signal board (Qy.0 and Qy.1 for PTO1 and Qy.2 and Qy.3 for PTO2)

Drive interface

For motion control, you can optionally configure a drive interface for "Drive enabled" and "Drive ready". When using the drive interface, the digital output for the drive enable and the digital input for "drive ready" can be freely selected.

Note

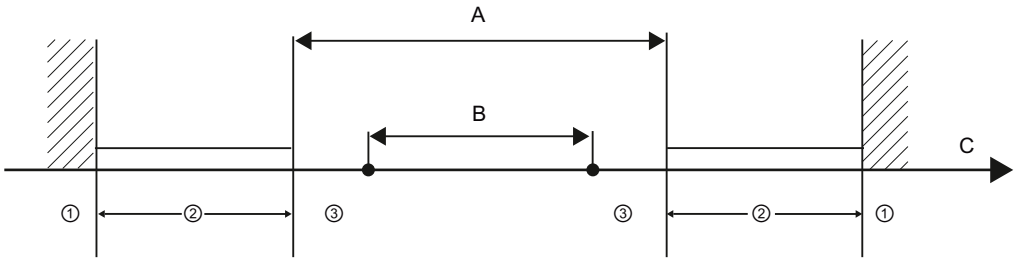
The firmware will take control via the corresponding pulse and direction outputs if the PTO (Pulse Train Output) has been selected and assigned to an axis.

With this takeover of the control function, the connection between the process image and I/O output is also disconnected. While the user has the possibility of writing the process image of pulse and direction outputs via the user program or watch table, this is never transferred to the I/O output. Accordingly, it is also not possible to monitor the I/O output via the user program or watch table. The information read merely reflects the value of the process image and does not match the actual status of the I/O output in any respect.

For all other CPU outputs that are not used permanently by the CPU firmware, the status of the I/O output can be controlled or monitored via the process image, as usual.

9.3.3.2 Hardware and software limit switches for motion control

Use the hardware and software limit switches to limit the "allowed travel range" and the "working range" of your axis.



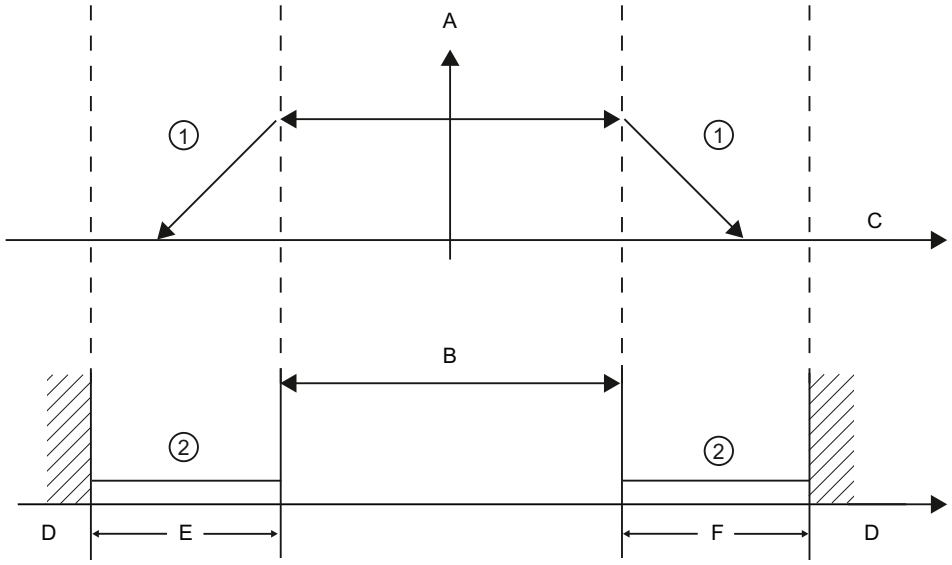
①	Mechanical stop	A	Allowed travel range for the axis
②	Lower and upper hardware limits	B	Working range of the axis
③	Lower and upper software limits	C	Distance

Hardware and software limit switches must be activated prior to use in the configuration or in the user program. Software limit switches are only active after homing the axis.

Hardware limit switches

Hardware limit switches determine the maximum travel range of the axis. Hardware limit switches are physical switching elements that must be connected to interrupt-capable inputs of the CPU. Use only hardware limit switches that remain permanently switched after being approached. This switching status may only be revoked after a return to the allowed travel range.

When the hardware limit switches are approached, the axis brakes to a standstill at the configured emergency deceleration. The specified emergency deceleration must be sufficient to reliably stop the axis before the mechanical stop. The following diagram presents the behavior of the axis after it approaches the hardware limit switches.

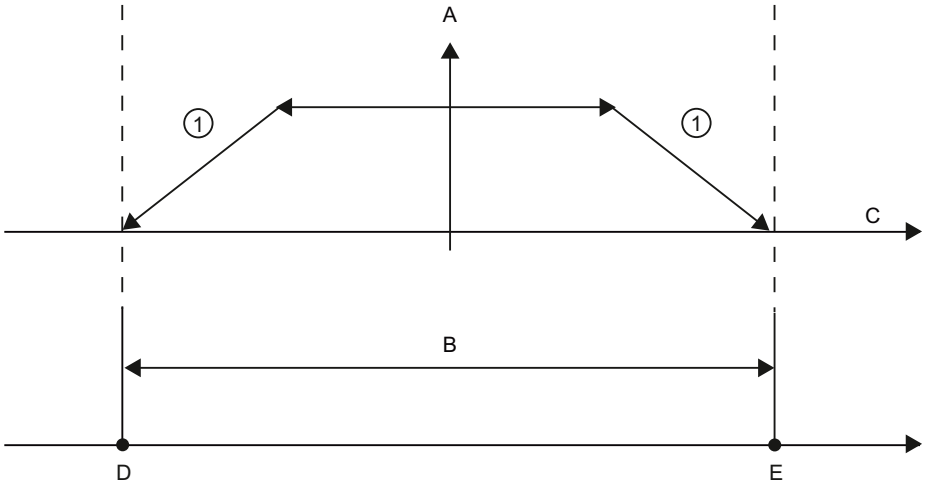


① The axis brakes to a standstill at the configured emergency decleration.
② Range in which the hardware limit switches signal the stats "approached".
A [Velocity]
B Allowed travel range
C Distance
D Mechanical stop
E Lower hardware limit switch
F Upper hardware limit switch

Software limit switches

Software limit switches limit the "working range" of the axis. They should fall inside the hardware limit switches relative to the travel range. Because the positions of the software limit switches can be set flexibly, the working range of the axis can be restricted on an individual basis depending on the current traversing profile. In contrast to hardware limit switches, software limit switches are implemented exclusively by means of the software and do not require their own switching elements.

If software limit switches are activated, an active motion is stopped at the position of the software limit switch. The axis is braked at the configured deceleration. The following diagram presents the behavior of the axis until it reaches the software limit switches.



① The axis brakes to a standstill at the configured deceleration.

A [Velocity]

B Working range

C Distance

D Lower software limit switch

E Upper software limit switch

Use additional hardware limit switches if a mechanical endstop is located after the software limit switches and there is a risk of mechanical damage.

Additional information

Your user program can override the hardware or software position limits by enabling or disabling both hardware and software limits functionality. The selection is made from the Axis DB.

- To enable or disable the hardware limit functionality, access the "Active" tag (Bool) in the DB path "<axis name>/Config/**PositonLimits_HW**". The state of the "Active" tag enables or disables the use of hardware position limits.
- To enable or disable software position limit functionality, access "Active" tag (Bool) in the DB path "<axis name>/Config/**Position Limits_SW**". The state of this "Active" tag enables or disables the software position limits.

You can also modify the software position limits with your user program (for example, to add flexibility for machine setup or to shorten machine change-over time). Your user program can write new values to the " MinPosition " and " MaxPosition " tags (engineering units in Real format) in the DB "<axis name>/Config/**PositionLimits_SW**".

9.3.3.3 Homing

Homing refers to the matching of the axis coordinates to the real, physical drive position. (If the drive is currently at position x, the axis will be adjusted to be in position x.) For position-controlled axes, the entries and displays for the position refer exactly to these axis coordinates.

Note

The agreement between the axis coordinates and the real situation is extremely important. This step is necessary to ensure that the absolute target position of the axis is also achieved exactly with the drive.

The MC_Home instruction initiates the homing of the axis.

There are 4 different homing functions. The first two functions allow the user to set the current position of the axis and the second two position the axis with respect to a Home reference Sensor.

- Mode 0 - Direct Referencing Absolute: When executed this mode tells the axis exactly where it is. It sets the internal position variable to the value of the Position input of the Homing instruction. This is used for machine calibration and setup.

  The axis position is set regardless of the reference point switch. Active traversing motions are not aborted. The value of the Position input parameter of the MC_Home instruction is set immediately as the reference point of the axis. To assign the reference point to an exact mechanical position, the axis must be at a standstill at this position at the time of the homing operation.

- Mode 1 - Direct Referencing Relative: When executed this mode uses the internal position variable and adds the value of the Position input on the Homing instruction to it. This is typically used to account for machine offset.

  The axis position is set regardless of the reference point switch. Active traversing motions are not aborted. The following statement applies to the axis position after homing: New axis position = current axis position + value of the Position parameter of the MC_Home instruction.

- Mode 2 - Passive Referencing: When the axis is moving and passes the Reference Point Switch the current position is set as the home position. This feature will help account for normal machine wear and gear backlash and prevent the need for manual compensation for wear. The Position input on the Homing instruction, as before, adds to the location indicated by the Reference Point Switch allowing easy offset of the Home position.

  During passive homing, the MC_Home instruction does not carry out any homing motion. The traversing motion required for this step must be implemented by the user via other motion control instructions. When the reference point switch is detected, the axis is homed according to the configuration. Active traversing motions are not aborted upon start of passive homing.

- Mode 3 - Active Referencing: This mode is the most precise method of Homing the Axis. The initial direction and velocity of movement is configured in the Technology Object Configuration Extended Parameters-Homing. This is dependent upon machine configuration. There is also the ability to determine if the leading edge or falling edge of the Reference Point Switch signal is the Home position. Virtually all sensors have an active range and if the Steady State On position was used as the Home signal then there would be a possibility for error in the Homing position since the On signal active range would cover a range of distance. By using either the leading or falling edge of that signal a much more precise Home position results. As with all other modes the value of the Position input on the Homing instruction is added to the Hardware referenced position.

  In active homing mode, the MC_Home instruction performs the required reference point approach. When the reference point switch is detected, the axis is homed according to the configuration. Active traversing motions are aborted.

Modes 0 and 1 do not require that the axis be moved at all. They are typically used in setup and calibration. Modes 2 and 3 require that the axis move and pass a sensor that is configured in the "Axis" technology object as the Reference Point Switch. The reference point which can be placed in the work area of the axis or outside of the normal work area but within movement range.

Configuration of homing parameters

Configure the parameters for active and passive homing in the "Homing" configuration window. The homing method is set using the "Mode" input parameter of the motion control instruction. Here, Mode = 2 means passive homing and Mode = 3 means active homing.

NOTICE
Use one of the following measures to ensure that the machine does not travel to a mechanical endstop in the event of a direction reversal: • Keep the approach velocity low • Increase the configured acceleration/deceleration • Increase the distance between hardware limit switch and mechanical stop

Table 9- 47 Configuration parameters for homing the axis

Parameter	Description
Input reference point switch (Active and passive homing)	Select the digital input for the reference point switch from the drop-down list box. The input must be interrupt-capable. The onboard CPU inputs and inputs of an inserted signal board can be selected as inputs for the reference point switch. The default filter time for the digital inputs is 6.4 ms. When the digital inputs are used as a reference point switch, this can result in undesired decelerations and thus inaccuracies. Depending on the reduced velocity and extent of the reference point switch, the reference point may not be detected. The filter time can be set under "Input filter" in the device configuration of the digital inputs. The specified filter time must be less than the duration of the input signal at the reference point switch.
Auto reverse after reaching the hardware limit switches (Active homing only)	Activate the check box to use the hardware limit switch as a reversing cam for the reference point approach. The hardware limit switches must be configured and activated for direction reversal. If the hardware limit switch is reached during active homing, the axis brakes at the configured deceleration (not with the emergency deceleration) and reverses direction. The reference point switch is then sensed in reverse direction. If the direction reversal is not active and the axis reaches the hardware limit switch during active homing, the reference point approach is aborted with an error and the axis is braked at the emergency deceleration.
Approach direction (Active and passive homing)	With the direction selection, you determine the "approach direction" used during active homing to search for the reference point switch, as well as the homing direction. The homing direction specifies the travel direction the axis uses to approach the configured side of the reference point switch to carry out the homing operation.
Reference point switch (Active and passive homing)	• Active homing: Select whether the axis is to be referenced on the left or right side of the reference point switch. Depending on the start position of the axis and the configuration of the homing parameters, the reference point approach sequence can differ from the diagram in the configuration window. • Passive homing: With passive homing, the traversing motions for purposes of homing must be implemented by the user via motion commands. The side of the reference point switch on which homing occurs depends on the following factors: – "Approach direction" configuration – "Reference point switch" configuration – Current travel direction during passive homing
Approach velocity (Active homing only)	Specify the velocity at which the reference point switch is to be searched for during the reference point approach. Limit values (independent of the selected user unit): Start/stop velocity ≤ approach velocity ≤ maximum velocity

Parameter	Description
Reduced velocity (Active homing only)	Specify the velocity at which the axis approaches the reference point switch for homing. Limit values (independent of the selected user unit): Start/stop velocity ≤ reduced velocity ≤ maximum velocity
Home position offset (Active homing only)	If the desired reference position deviates from the position of the reference point switch, the home position offset can be specified in this field. If the value does not equal 0, the axis executes the following actions following homing at the reference point switch: 1. Move the axis at reduced velocity by the value of the home position offset. 2. When the position of the home position offset is reached, the axis position is set to the absolute reference position. The absolute reference position is specified via parameter "Position" of motion control instruction "MC_Home". Limit values (independent of the selected user unit): -1.0e12 ≤ home position offset ≤ 1.0e12

Table 9- 48 Factors that affect homing

Influencing factors:			Result:
Configuration Approach direction	Configuration Reference point switch	Current travel direction	Homing on Reference point switch
Positive	"Left (negative) side"	Positive direction	Left
		Negative direction	Right
Positive	"Right (positive) side"	Positive direction	Right
		Negative direction	Left
Negative	"Left (negative) side"	Positive direction	Right
		Negative direction	Left
Negative	"Right (positive) side"	Positive direction	Left
		Negative direction	Right

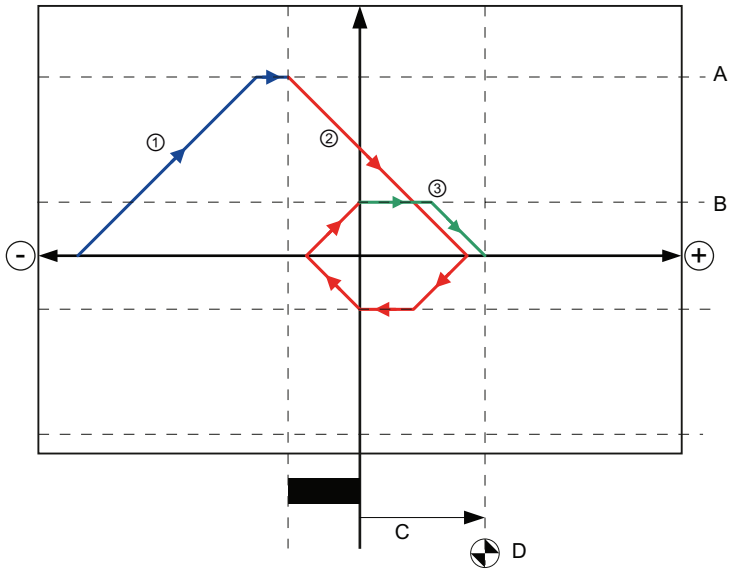
Sequence for active homing

You start active homing with motion control instruction "MC_Home" (input parameter Mode = 3). Input parameter "Position" specifies the absolute reference point coordinates in this case. Alternatively, you can start active homing on the control panel for test purposes.

The following diagram shows an example of a characteristic curve for an active reference point approach with the following configuration parameters:

- "Approach direction" = "Positive approach direction"
- "Reference point switch" = "Right (positive) side"
- Value of "home position offset" > 0

Table 9- 49 Velocity characteristics of MC homing

Operation	Notes	
	A	Approach velocity
	B	Reduced velocity
	C	Home position coordinate
	D	Home position offset

①	Search phase (blue curve segment): When active homing starts, the axis accelerates to the configured "approach velocity" and searches at this velocity for the reference point switch.
②	Reference point approach (red curve section): When the reference point switch is detected, the axis in this example brakes and reverses, to be homed on the configured side of the reference point switch at the configured "reduced velocity".
③	Travel to reference point position (green curve segment): After homing at the reference point switch, the axis travels to the "Reference point coordinates" at the "reduced velocity". On reaching the "Reference point coordinates", the axis is stopped at the position value that was specified in the Position input parameter of the MC_Home instruction".

Note

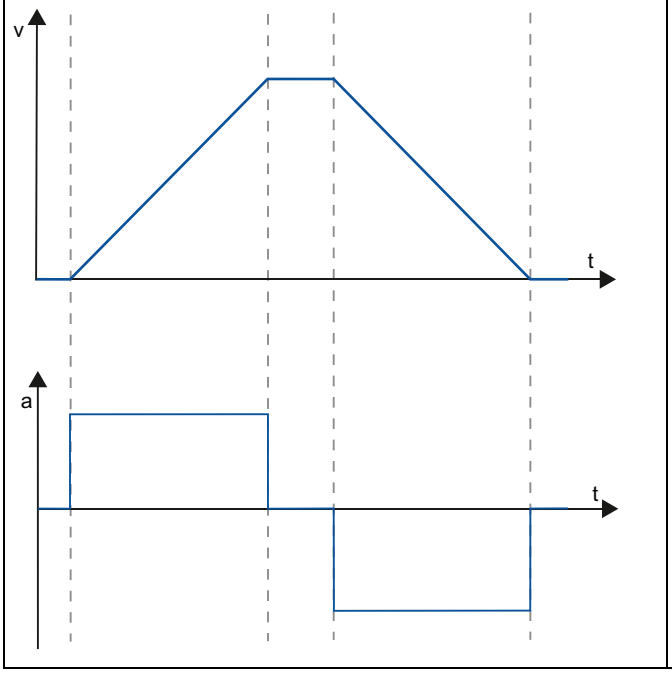
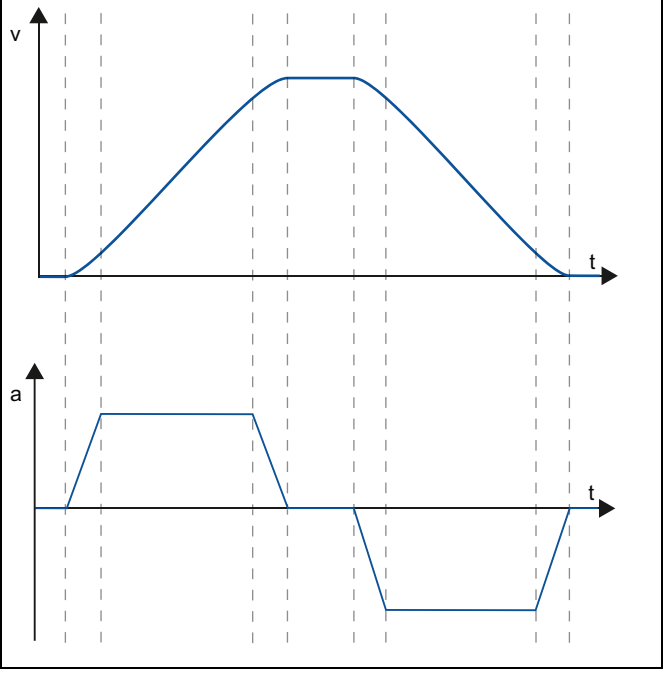
If the homing search does not function as you expected, check the inputs assigned to the hardware limits or to the reference point. These inputs may have had their edge interrupts disabled in device configuration.

Examine the configuration data for the axis technology object of concern to see which inputs (if any) are assigned for "HW Low Limit Switch Input", "HW High Limit Switch Input", and "Input reference point switch". Then open the Device configuration for the CPU and examine each of the assigned inputs. Verify the "Enable rising edge detection" and "Enable falling edge detection" are both selected. If these properties are not selected, delete the specified inputs in the axis configuration and select them again.

9.3.3.4 Jerk limit

With the jerk limit you can reduce the stresses on your mechanics during an acceleration and deceleration ramp. The value for the acceleration and deceleration is not changed abruptly when the step limiter is active; it is adapted in a transition phase. The figure below shows the velocity and acceleration curve without and with jerk limit.

Table 9- 50 Jerk limit

Travel without step limiter	Travel with step limiter
	

The jerk limit gives a "smoothed" velocity profile of the axis motion. This ensures soft starting and braking of a conveyor belt for example.

9.3.4 Commissioning

"Status and error bits" diagnostic function

Use the "Status and error bits" diagnostic function to monitor the most important status and error messages for the axis. The diagnostic function display is available in online mode in "Manual control" mode and in "Automatic control" when the axis is active.

Table 9- 51 Status of the axis

Status	Description
Enabled	The axis is enabled and ready to be controlled via motion control tasks. (Tag of technology object: <Axis name>.StatusBits.Enable)
Homed	The axis is homed and is capable of executing absolute positioning tasks of motion control instruction "MC_MoveAbsolute". The axis does not have to be homed for relative homing. Special situations: • During active homing, the status is FALSE. • If a homed axis undergoes passive homing, the status is set to TRUE during passive homing. (Tag of technology object: <Axis name>.StatusBits.HomingDone)
Error	An error has occurred in the "Axis" technology object. More information about the error is available in automatic control at the ErrorID and ErrorInfo parameters of the motion control instructions. In manual mode, the "Last error" field of the control panel displays detailed information about the cause of error. (Tag of technology object: <Axis name>.StatusBits.Error)
Control panel active	The "Manual control" mode was enabled in the control panel. The control panel has control priority over the "Axis" technology object. The axis cannot be controlled from the user program. (Tag of technology object: <Axis name>.StatusBits.ControlPanelActive)

Table 9- 52 Drive status

Status	Description
Drive ready	The drive is ready for operation. (Tag of technology object: <Axis name>.StatusBits.DriveReady)
Error	The drive has reported an error after failure of its ready signal. (Tag of technology object: <Axis name>.ErrorBits.DriveFault)

Table 9- 53 Status of the axis motion

Status	Description
Standstill	The axis is at a standstill. (Tag of technology object: <Axis name>.StatusBits.StandStill)
Accelerating	The axis accelerates. (Tag of technology object: <Axis name>.StatusBits.Acceleration)

Status	Description
Constant velocity	The axis travels at constant velocity. (Tag of technology object: <Axis name>.StatusBits.ConstantVelocity)
Decelerating	The axis decelerates (slows down). (Tag of technology object: <Axis name>.StatusBits.Deceleration)

Table 9- 54 Status of the motion mode

Status	Description
Positioning	The axis executes a positioning task of motion control instruction "MC_MoveAbsolute" or "MC_MoveRelative" or of the control panel. (Tag of technology object: <Axis name>.StatusBits.PositioningCommand)
Speed Command	The axis executes a task at set speed of motion control instruction "MC_MoveVelocity" or "MC_MoveJog" or of the control panel. (Tag of technology object: <Axis name>.StatusBits.SpeedCommand)
Homing	The axis executes a homing task of motion control instruction "MC_Home" or the control panel. (Tag of technology object: <Axis name>.StatusBits.Homing)

Table 9- 55 Error bits

Error	Description
Min software limit reached	The lower software limit switch has been reached. (Tag of technology object: <Axis name>.ErrorBits.SwLimitMinReached)
Min software limit exceeded	The lower software limit switch has been exceeded. (Tag of technology object: <Axis name>.ErrorBits.SwLimitMinExceeded)
Max software limit reached	The upper software limit switch has been reached. (Tag of technology object: <Axis name>.ErrorBits.SwLimitMaxReached)
Max software limit exceeded	The upper software limit switch has been exceeded. (Tag of technology object: <Axis name>.ErrorBits.SwLimitMaxExceeded)
Negative hardware limit	The lower hardware limit switch has been approached. (Tag of technology object: <Axis name>.ErrorBits.HwLimitMin)
Positive hardware limit	The upper hardware limit switch has been approached. (Tag of technology object: <Axis name>.ErrorBits.HwLimitMax)
PTO and HSC already used	A second axis is using the same PTO and HSC and is enabled with "MC_Power". (Tag of technology object: <Axis name>.ErrorBits.HwUsed)
Configuration error	The "Axis" technology object was incorrectly configured or editable configuration data were modified incorrectly during runtime of the user program. (Tag of technology object: <Axis name>.ErrorBits.ConfigFault)
General Error	An internal error has occurred. (Tag of technology object: <Axis name>.ErrorBits.SystemFault)

"Motion status" diagnostic function

Use the "Motion status" diagnostic function to monitor the motion status of the axis. The diagnostic function display is available in online mode in "Manual control" mode and in "Automatic control" when the axis is active.

Table 9- 56 Motion status

Status	Description
Target position	The "Target position" field indicates the current target position of an active positioning task of motion control instruction "MC_MoveAbsolute" or "MC_MoveRelative" or of the control panel. The value of the "Target position" is only valid during execution of a positioning task. (Tag of technology object: <Axis name>.MotionStatus.TargetPosition)
Current position	The "Current position" field indicates the current axis position. If the axis is not homed, the value indicates the position value relative to the enable position of the axis. (Tag of technology object: <Axis name>.MotionStatus.Position)
Current velocity	The "Current velocity" field indicates the actual axis velocity. (Tag of technology object: <Axis name>.MotionStatus.Velocity)

Table 9- 57 Dynamic limits

Dynamic limit	Description
Velocity	The "Velocity" field indicates the configured maximum velocity of the axis. (Tag of technology object: <Axis name>.Config.DynamicLimits.MaxVelocity)
Acceleration	The "Acceleration" field indicates the currently configured acceleration of the axis. (Tag of technology object: <Axis name>.Config.DynamicDefaults.Acceleration)
Deceleration	The "Deceleration" field indicates the currently configured deceleration of the axis. (Tag of technology object: <Axis name>.Config.DynamicDefaults.Deceleration)

9.3.5 Monitoring active commands

9.3.5.1 Monitoring MC instructions with a "Done" output parameter

Motion control instructions with the output parameter "Done" are started by the input parameter "Execute" and have a defined conclusion (for example, with motion control instruction "MC_Home": Homing was successful). The task is complete and the axis is at a standstill.

- The output parameter "Done" indicates the value TRUE, if the task has been successfully completed.
- The output parameters "Busy", "CommandAborted", and "Error" signal that the task is still being processed, has been aborted or an error is pending. The motion control instruction "MC_Reset" cannot be aborted and thus has no "CommandAborted" output parameter.
  - During processing of the motion control task, the output parameter "Busy" indicates the value TRUE. If the task has been completed, aborted, or stopped by an error, the output parameter "Busy" changes its value to FALSE. This change occurs regardless of the signal at input parameter "Execute".
  - Output parameters "Done", "CommandAborted", and "Error" indicate the value TRUE for at least one cycle. These status messages are latched while input parameter "Execute" is set to TRUE.

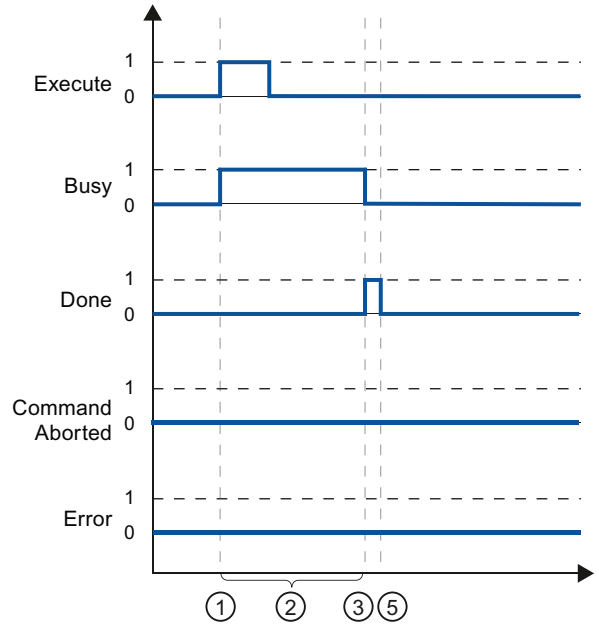
The tasks of the following motion control instructions have a defined conclusion:

- MC_Reset
- MC_Home
- MC_Halt
- MC_MoveAbsolute
- MC_MoveRelative

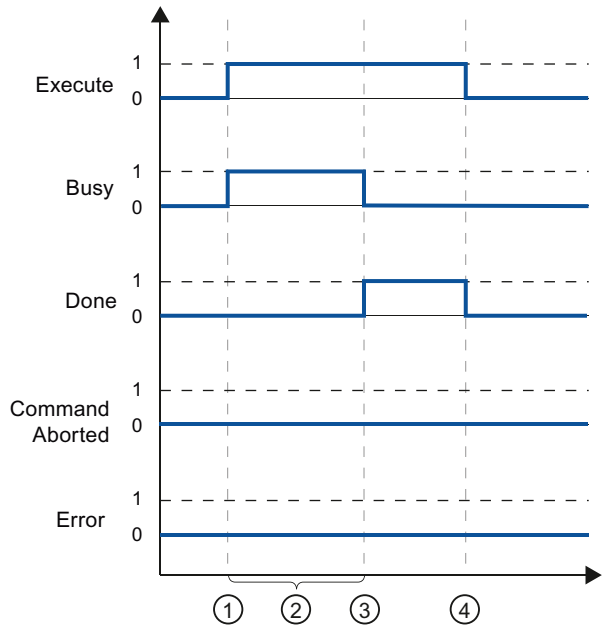
The behavior of the status bits is presented below for various example situations.

- The first example shows the behavior of the axis for a completed task. If the motion control task has been completely executed by the time of its conclusion, this is indicated by the value TRUE in output parameter "Done". The signal status of input parameter "Execute" influences the display duration in the output parameter "Done".
- The second example shows the behavior of the axis for an aborted task. If the motion control task is aborted during execution, this is indicated by the value TRUE in output parameter "CommandAborted". The signal status of the input parameter "Execute" influences the display duration in the output parameter "CommandAborted".
- The third example shows the behavior of the axis if an error occurs. If an error occurs during execution of the motion control task, this is indicated by the value TRUE in the output parameter "Error". The signal status of the input parameter "Execute" influences the display duration in the output parameter "Error".

Table 9- 58 Example 1 - Complete execution of task



If "Execute" = FALSE during the processing of the task



If "Execute" = FALSE after completion of the task

① The task is started with a positive edge at the input parameter "Execute". Depending on the programming, "Execute" can still be reset to the value FALSE during the task, or the value TRUE can be retained until after completion of the task.

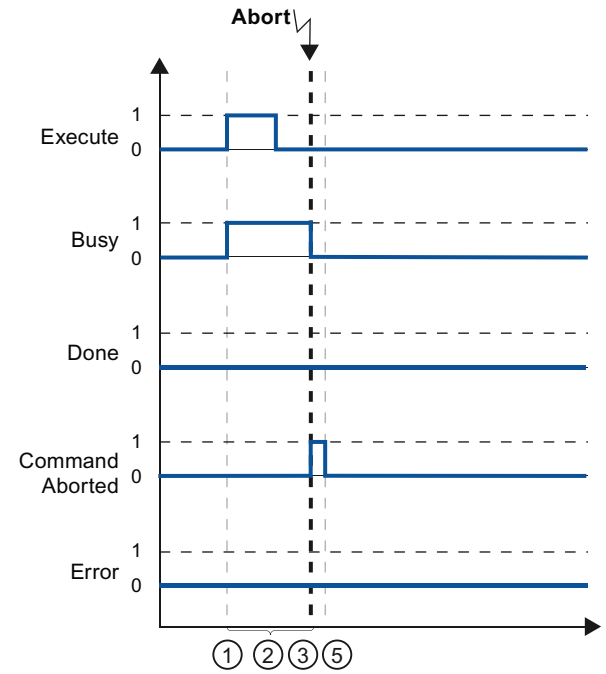
② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ With conclusion of the task (for example, for motion control instruction "MC_Home": Homing was successful), output parameter "Busy" changes to FALSE and "Done" to TRUE.

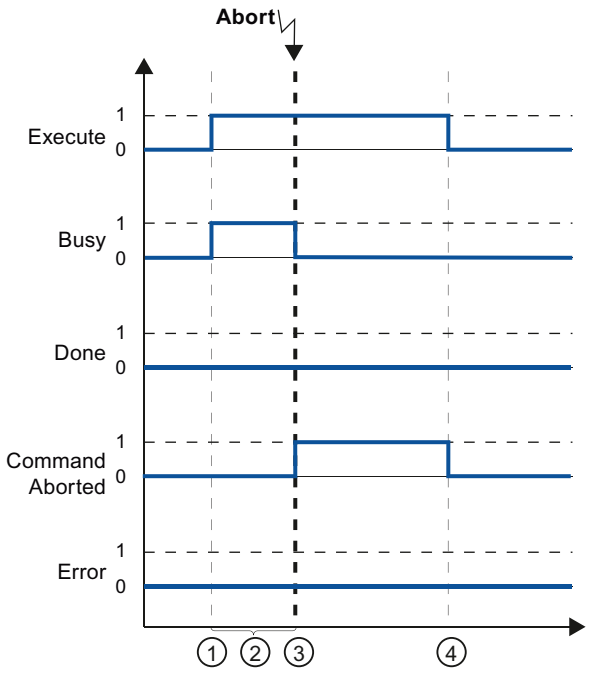
④ If "Execute" retains the value TRUE until after completion of the task, then "Done" also remains TRUE and changes its value to FALSE together with "Execute".

⑤ If "Execute" has been set to FALSE before the task is complete, "Done" indicates the value TRUE for only one execution cycle.

Table 9- 59 Example 2 - Aborting the task



If "Execute" = FALSE before the task is aborted



If "Execute" = FALSE after the task is aborted

① The task is started with a positive edge at the input parameter "Execute". Depending on the programming, "Execute" can still be reset to the value FALSE during the task, or the value TRUE can be retained until after completion of the task.

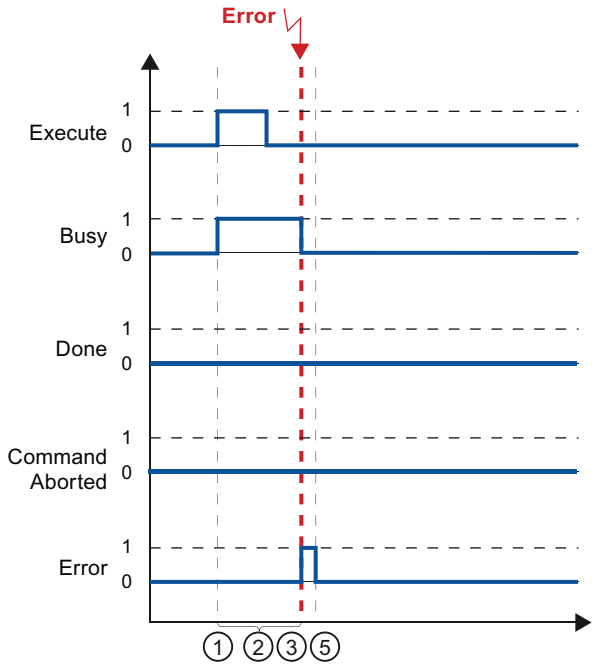
② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ During task execution, the task is aborted by another motion control task. If the task is aborted, output parameter "Busy" changes to FALSE and "CommandAborted" to TRUE.

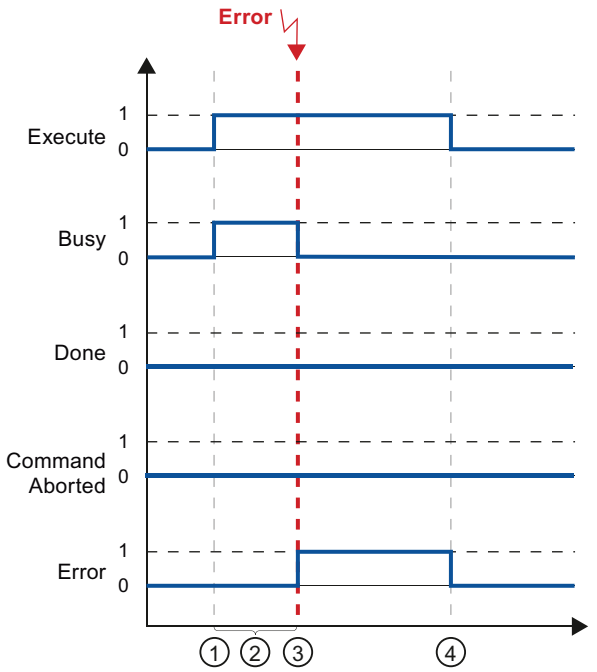
④ If "Execute" retains the value TRUE until after the task is aborted, then "CommandAborted" also remains TRUE and changes its value to FALSE together with "Execute".

⑤ If "Execute" has been set to FALSE before the task is aborted, "CommandAborted" indicates the value TRUE for only one execution cycle.

Table 9- 60 Example 3 - Error during task execution



If "Execute" = FALSE before the error occurs



If "Execute" = FALSE after the error occurs

① The task is started with a positive edge at the input parameter "Execute". Depending on the programming, "Execute" can still be reset to the value FALSE during the task, or the value TRUE can be retained until after completion of the task.

② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ An error occurred during task execution. When the error occurs, the output parameter "Busy" changes to FALSE and "Error" to TRUE.

④ If "Execute" retains the value TRUE until after the error occurs, then "Error" also remains TRUE and only changes its value to FALSE together with "Execute".

⑤ If "Execute" has been set to FALSE before the error occurs, "Error" indicates the value TRUE for only one execution cycle.

9.3.5.2 Monitoring the MC_Velocity instruction

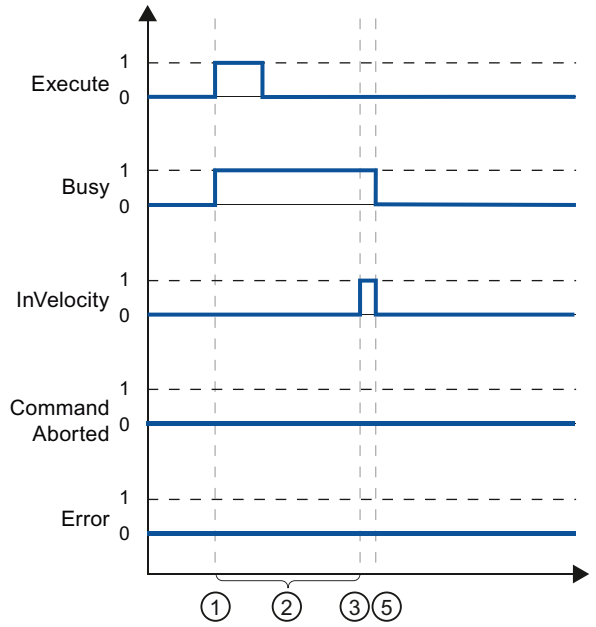
The tasks of motion control instruction "MC_MoveVelocity" constantly at the specified velocity.

- The tasks of motion control instruction "MC_MoveVelocity" do not have a defined end. The task objective is fulfilled when the parameterized velocity is reached for the first time and the axis travels at constant velocity. When the parameterized velocity is reached, this is indicated by the value TRUE in output parameter "InVelocity".
- The task is complete when the parameterized velocity has been reached and input parameter "Execute" has been set to the value FALSE. However, the axis motion is not yet complete upon completion of the task. For example, the axis motion can be stopped with motion control task "MC_Halt".
- The output parameters "Busy", "CommandAborted", and "Error" signal that the task is still being processed, has been aborted or an error is pending.
  - During execution of the motion control task, output parameter "Busy" indicates the value TRUE. If the task has been completed, aborted, or stopped by an error, the output parameter "Busy" changes its value to FALSE. This change occurs regardless of the signal at input parameter "Execute".
  - The output parameters "InVelocity", "CommandAborted", and "Error" indicate the value TRUE for at least one cycle, when their conditions are met. These status messages are latched while input parameter "Execute" is set to TRUE.

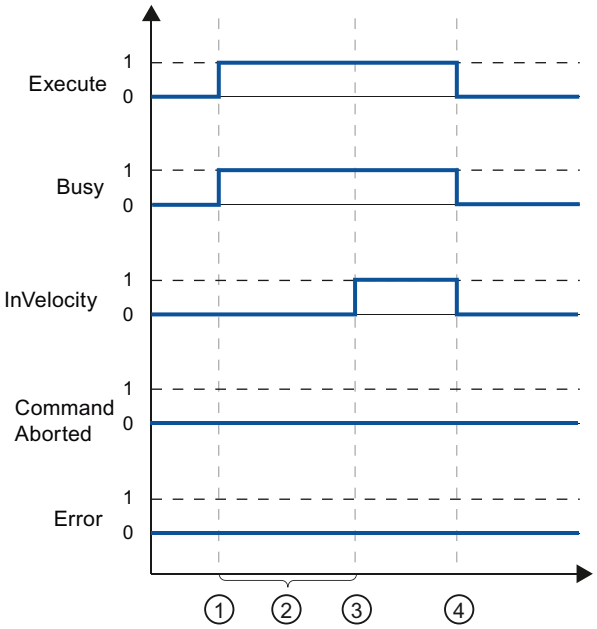
The behavior of the status bits is presented below for various example situations.

- The first example shows the behavior when the axis reaches the parameterized velocity. If the motion control task has been executed by the time the parameterized velocity is reached, this is indicated by the value TRUE in output parameter "InVelocity". The signal status of the input parameter "Execute" influences the display duration in the output parameter "InVelocity".
- The second example shows the behavior if the task is aborted before achieving the parameterized velocity. If the motion control task is aborted before the parameterized velocity is reached, this is indicated by the value TRUE in output parameter "CommandAborted". The signal status of input parameter "Execute" influences the display duration in output parameter "CommandAborted".
- The third example shows the behavior of the axis if an error occurs before achieving the parameterized velocity. If an error occurs during execution of the motion control task before the parameterized velocity has been reached, this is indicated by the value TRUE in the output parameter "Error". The signal status of the input parameter "Execute" influences the display duration in the output parameter "Error".

Table 9- 61 Example 1 - If the parameterized velocity is reached



If "Execute" = FALSE before the configured velocity is reached



If "Execute" = FALSE after the configured velocity is reached

① The task is started with a positive edge at the input parameter "Execute". Depending on the programming, "Execute" can be reset to the value FALSE event before the parameterized velocity is reached, or alternatively only after it has been reached.

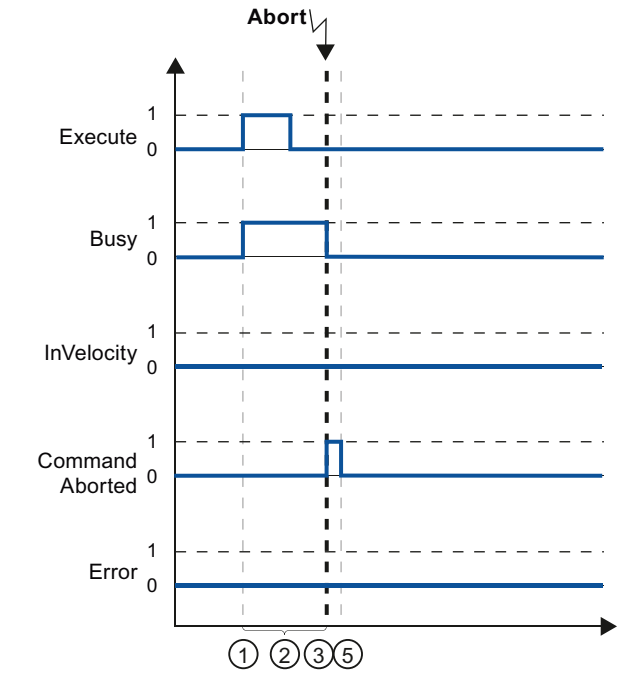
② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ When the parameterized velocity is reached, the output parameter "InVelocity" changes to TRUE.

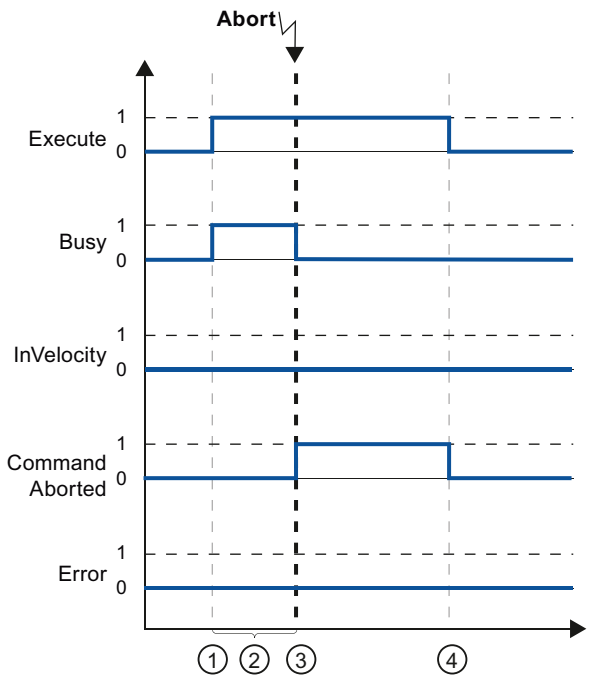
④ If "Execute" retains the value TRUE even after the parameterized velocity has been reached, the task remains active. "InVelocity" and "Busy" retain the value TRUE and only change their status to FALSE together with "Execute".

⑤ If "Execute" has been reset to FALSE before the parameterized velocity is reached, the task is complete when the parameterized velocity is reached. "InVelocity" indicates the value TRUE for one execution cycle and changes to FALSE together with "Busy".

Table 9- 62 Example 2 - If the task is aborted prior to reaching the parameterized velocity



If "Execute" = FALSE before the task is aborted



If "Execute" = FALSE after the task is aborted

① The task is started with a positive edge at the input parameter "Execute". Depending on the programming, "Execute" can still be reset to the value FALSE during the task, or the value TRUE can be retained until after the task is aborted.

② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ During task execution, the task is aborted by another motion control task. If the task is aborted, output parameter "Busy" changes to FALSE and "CommandAborted" to TRUE.

④ If "Execute" retains the value TRUE until after the task is aborted, then "CommandAborted" also remains TRUE and changes its status to FALSE together with "Execute".

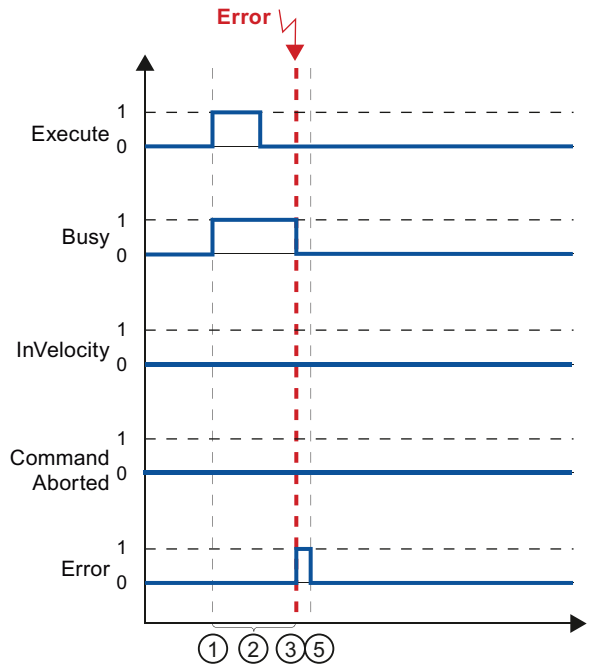
⑤ If "Execute" has been reset to FALSE before the task is aborted, "CommandAborted" indicates the value TRUE for only one execution cycle.

Note

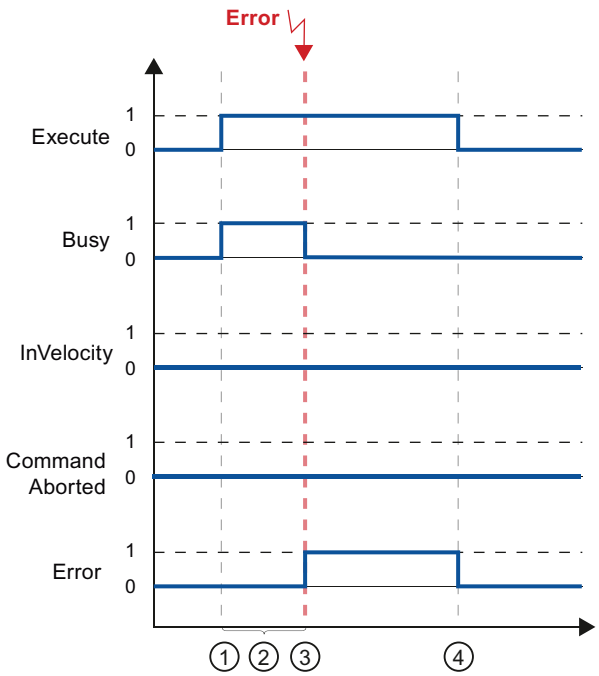
Under the following conditions, an abort is not indicated in output parameter "CommandAborted":

- The parameterized velocity has been reached, input parameter "Execute" has the value FALSE, and a new motion control task is initiated.
- When the parameterized velocity is reached and input parameter "Execute" has the value FALSE, the task is complete. Therefore, the start of a new task is not indicated as an abort.

Table 9- 63 Example 3 - If an error occurs prior to reaching the parameterized velocity



If "Execute" = FALSE before the error occurs



If "Execute" = FALSE after the error occurs

① The task is started with a positive edge at the input parameter "Execute". Depending on the programming, "Execute" can still be reset to the value FALSE during the task, or the value TRUE can be retained until after the error has occurred.

② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ An error occurred during task execution. When the error occurs, the output parameter "Busy" changes to FALSE and "Error" to TRUE.

④ If "Execute" retains the value TRUE until after the error has occurred, then "Error" also remains TRUE and only changes its status to FALSE together with "Execute".

⑤ If "Execute" has been reset to FALSE before the error occurs, "Error" indicates the value TRUE for only one execution cycle.

Note

Under the following conditions, an error is not indicated in output parameter "Error":

- The parameterized velocity has been reached, input parameter "Execute" has the value FALSE, and an axis error occurs (software limit switch is approached, for example).
- When the parameterized velocity is reached and input parameter "Execute" has the value FALSE, the task is complete. After completion of the task, the axis error is only indicated in the motion control instruction "MC_Power".

9.3.5.3 Monitoring the MC_MoveJog instruction

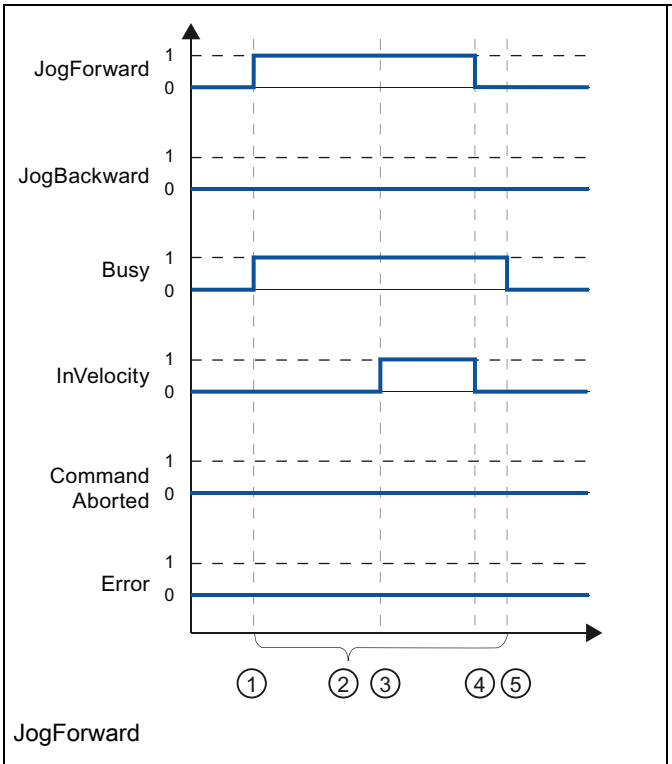
The tasks of motion control instruction "MC_MoveJog" implement a jog operation.

- The motion control tasks "MC_MoveJog" do not have a defined end. The task objective is fulfilled when the parameterized velocity is reached for the first time and the axis travels at constant velocity. When the parameterized velocity is reached, this is indicated by the value TRUE in output parameter "InVelocity".
- The order is complete when input parameter "JogForward" or "JogBackward" has been set to the value FALSE and the axis has come to a standstill.
- The output parameters "Busy", "CommandAborted", and "Error" signal that the task is still being processed, has been aborted or an error is pending.
  - During processing of the motion control task, the output parameter "Busy" indicates the value TRUE. If the task has been completed, aborted, or stopped by an error, the output parameter "Busy" changes its value to FALSE.
  - The output parameter "InVelocity" indicates the status TRUE, as long as the axis is moving at the parameterized velocity. The output parameters "CommandAborted" and "Error" indicate the status for at least one cycle. These status messages are latched as long as either input parameter "JogForward" or "JogBackward" is set to TRUE.

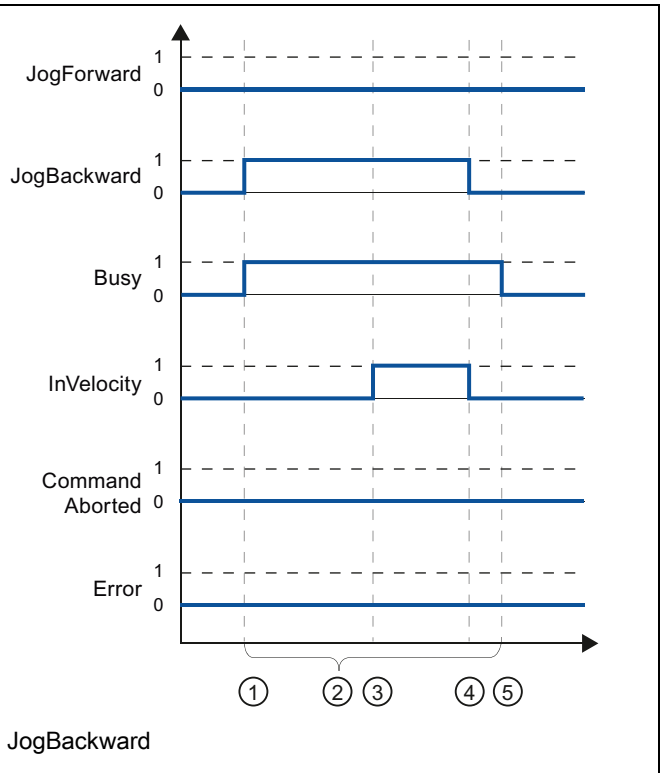
The behavior of the status bits is presented below for various example situations.

- The first example shows the behavior or the axis if the parameterized velocity is reached and maintained. If the motion control task has been executed by the time the parameterized velocity is reached, this is indicated by the value TRUE in output parameter "InVelocity".
- The second example shows the behavior of the axis if the task is aborted. If the motion control task is aborted during execution, this is indicated by the value TRUE in output parameter "CommandAborted". The behavior is independent of whether or not the parameterized velocity has been reached.
- The third example shows the behavior of the axis if an error occurs. If an error occurs during execution of the motion control task, this is indicated by the value TRUE in output parameter "Error". The behavior is independent of whether or not the parameterized velocity has been reached.

Table 9- 64 Example 1 - If the parameterized velocity is reached and maintained



JogForward



JogBackward

① The task is started with a positive edge at the input parameter "JogForward" or "JogBackward".

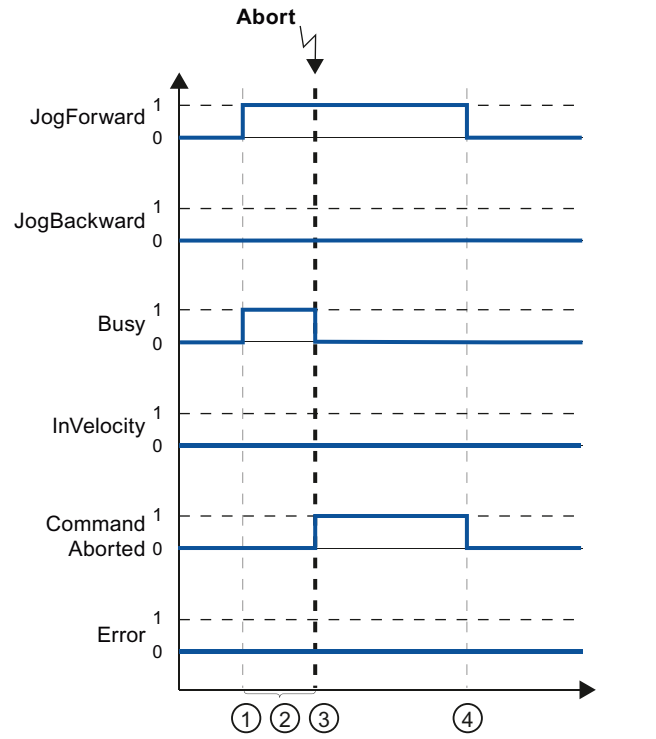
② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ When the parameterized velocity is reached, the output parameter "InVelocity" changes to TRUE.

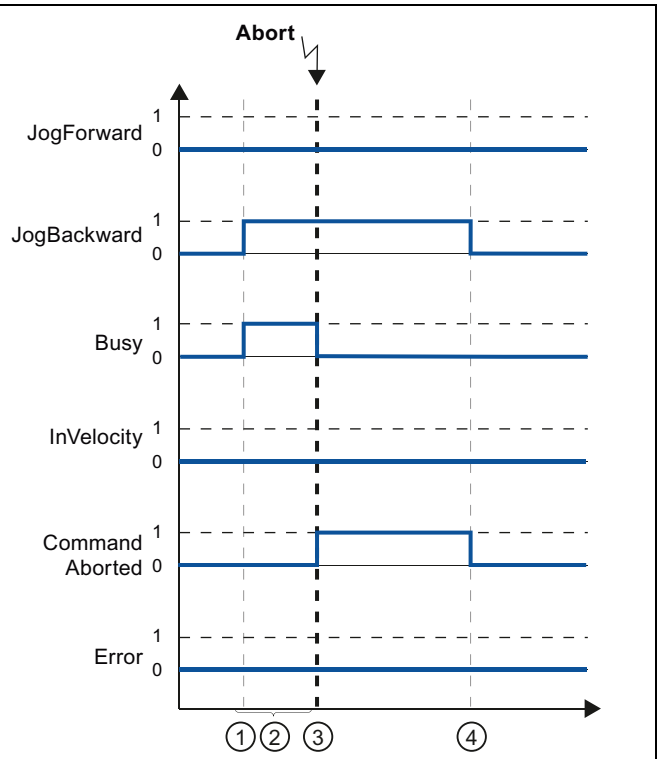
④ When the input parameter "JogForward" or "JogBackward" is reset to the value FALSE, the axis motion ends. The axis starts to decelerate. As a result, the axis no longer moves at constant velocity and the output parameter "InVelocity" changes its status to FALSE.

⑤ If the axis has come to a standstill, the motion control task is complete and the output parameter "Busy" changes its value to FALSE.

Table 9- 65 Example 2 - If the task is aborted during execution



JogForward



JogBackward

① The task is started with a positive edge at the input parameter "JogForward" or "JogBackward".

② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ During task execution, the task is aborted by another motion control task. If the task is aborted, output parameter "Busy" changes to FALSE and "CommandAborted" to TRUE.

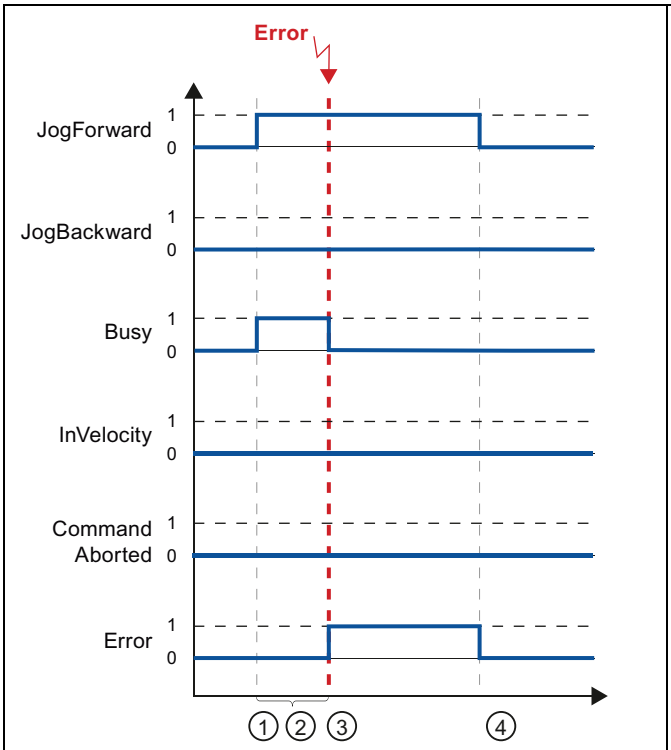
④ When the input parameter "JogForward" or "JogBackward" is reset to the value FALSE, the output parameter "CommandAborted" changes its value to FALSE.

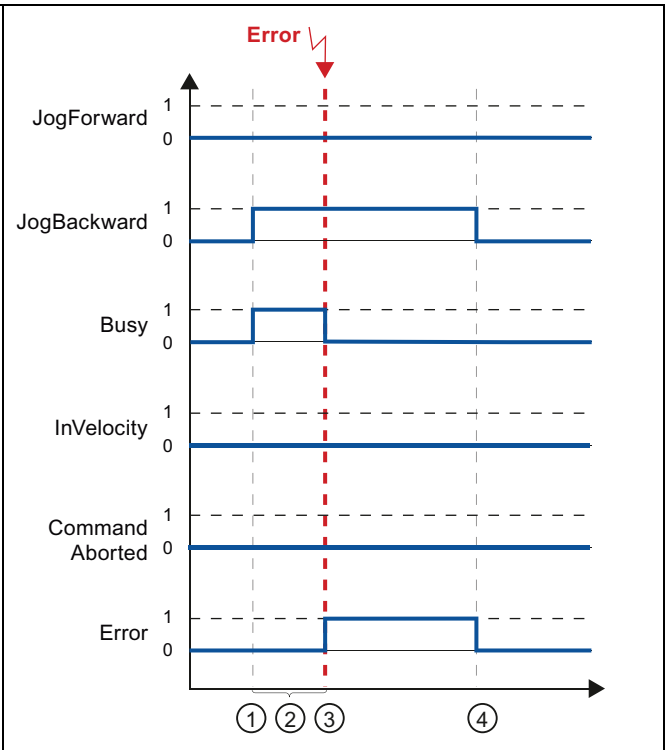
Note

The task abort is indicated in the output parameter "CommandAborted" for only one execution cycle, if all conditions below are met:

The input parameters "JogForward" and "JogBackward" have the value FALSE (but the axis is still decelerating) and a new motion control task is initiated.

Table 9- 66 Example 3 - If an error has occurred during task execution





① The task is started with a positive edge at the input parameter "JogForward" or "JogBackward".

② While the task is active, the output parameter "Busy" indicates the value TRUE.

③ An error occurred during task execution. When the error occurs, the output parameter "Busy" changes to FALSE and "Error" to TRUE.

④ When the input parameter "JogForward" or "JogBackward" is reset to the value FALSE, the output parameter "Error" changes its value to FALSE.

Note

An error occurrence is indicated in the output parameter "Error" for only one execution cycle, if all the conditions below are met:

The input parameters "JogForward" and "JogBackward" have the value FALSE (but the axis is still decelerating) and a new error occurs (software limit switch is approached, for example).

Communication 10

The S7-1200 offers several types of communication between CPUs and programming devices, HMIs, and other CPUs:

- PROFINET (for exchanging data through the user program with other communications partners via Ethernet):
  - For PROFINET and PROFIBUS, the CPU supports a total of 16 devices and 256 submodules, with a maximum of 8 PROFINET IO devices and 128 submodules (if eight or less PROFIBUS slaves or submodules are configured).
  - S7 communication
  - User Datagram Protocol (UDP) protocol
  - ISO on TCP (RFC 1006)
  - Transport Control Protocol (TCP)

As an IO controller using PROFINET RT, the S7-1200 communicates with up to 8 PN devices on the local PN network or through a PN/PN coupler (link). Refer to PROFIBUS and PROFINET International, PI (www.us.profinet.com) for more information.

- PROFIBUS:
  - CM 1242-5: Operates as DP slave
  - CM 1243-5: Operates as DP master class1
  - For PROFINET and PROFIBUS, the CPU supports a total of 16 devices and 256 submodules, with a maximum of 16 PROFIBUS DP slave devices and 256 submodules (if no PROFINET IO devices or submodules are configured).

    Note

    The 16 devices include the following:
    - The DP slave modules attached by the DP master (CM 1243-5)
    - Any DP slave module (CM 1242-5) connected to the CPU
    - Any PROFINET device connected to the CPU over the PROFINET port

    For example, a configuration with three PROFIBUS CMs (one CM 1243-5 master and two CM 1242-5 slave modules) would reduce the maximum number of slave modules that can be accessed by the DP Master (CM 1243-5) to 14.

- AS-i: The S7-1200 CM 1243-2 AS-i Master allows the attachment of an AS-i network to an S7-1200 CPU.
- CPU-to-CPU S7 communication
- Teleservice communication

10.1 Number of asynchronous communication connections supported

The CPU supports the following maximum number of simultaneous, asynchronous communication connections for PROFINET and PROFIBUS:

- 8 connections for Open User Communications (active or passive): TSEND_C, TRCV_C, TCON, TDISCON, TSEND, and TRCV.
- 3 CPU-to-CPU S7 connections for server GET/PUT data
- 8 CPU-to-CPU S7 connections for client GET/PUT data

Note

S7-1200, S7-300, and S7-400 CPUs use the GET and PUT instructions for CPU-to-CPU S7 communication. An S7-200 CPU uses ETHx_XFER instructions for CPU-to-CPU S7 communication.

- HMI connections: The CPU provides dedicated HMI connections to support up to 3 HMI devices. (You can have up to 2 SIMATIC Comfort panels.) The total number of HMI is affected by the types of HMI panels in your configuration. For example, you could have up to three SIMATIC Basic panels connected to your CPU, or you could have up to two SIMATIC Comfort panels with one additional Basic panel.
- PG connections: The CPU provides connections to support 1 programming device (PG).
- Webserver (HTTP) connections: The CPU provides connections for the Webserver.

10.2 PROFINET

10.2.1 Local/Partner connection

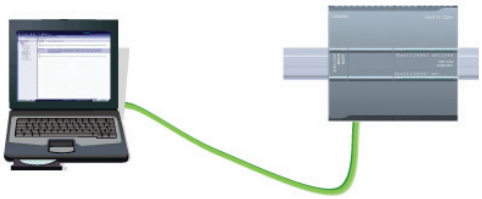
A Local / Partner (remote) connection defines a logical assignment of two communication partners to establish communication services. A connection defines the following:

- Communication partners involved (One active, one passive)
- Type of connection (for example, a PLC, HMI, or device connection)
- Connection path

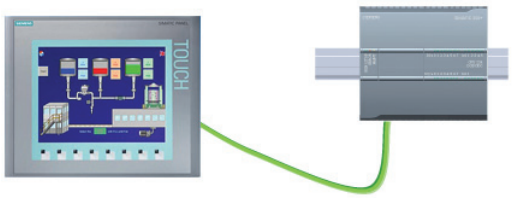
Communication partners execute the instructions to set up and establish the communication connection. You use parameters to specify the active and passive communication end point partners. After the connection is set up and established, it is automatically maintained and monitored by the CPU. Refer to the section on "Configuring the Local/Partner connection" (Page 118) for information about configuring the parameters for the connection.

If the connection is terminated (for example, due to a line break), the active partner attempts to re-establish the configured connection. You do not have to execute the communication instruction again.

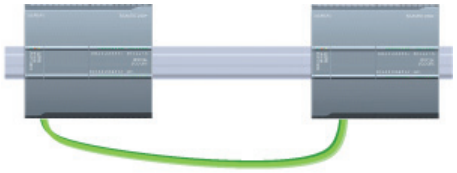
The CPU can communicate with other CPUs, with programming devices, with HMI devices, and with non-Siemens devices using standard TCP communications protocols.



Programming device connected to the CPU



HMI connected to the CPU



A CPU connected to another CPU

The PROFINET port on the CPU does not contain an Ethernet switching device. A direct connection between a programming device or HMI and a CPU does not require an Ethernet switch. However, a network with more than two CPUs or HMI devices requires an Ethernet switch.

① CSM1277 Ethernet switch

You can use the rack-mounted CSM1277 4-port Ethernet switch for connecting multiple CPUs and HMI devices.

10.2.2 Open user communication

10.2.2.1 Connection IDs for the PROFINET instructions

When you insert the TSEND_C, TRCV_C or TCON PROFINET instructions into your user program, STEP 7 creates an instance DB to configure the communications channel (or connection) between the devices (Page 118). Use the "Properties" of the instruction to configure the parameters for the connection. Among the parameters is the connection ID for that connection.

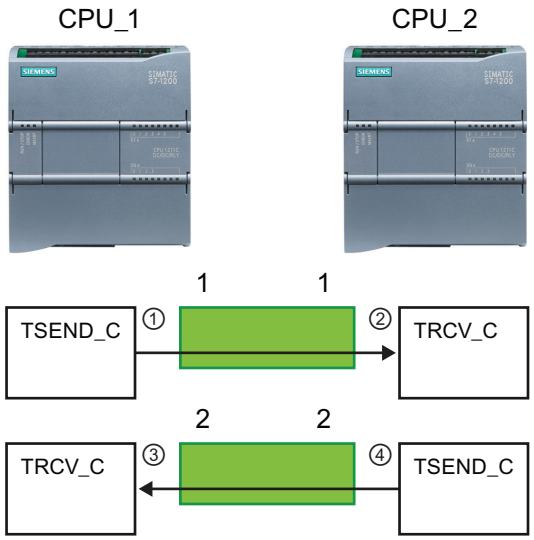
- The connection ID must be unique for the CPU. Each connection that you create must have a different DB and connection ID.
- Both the local CPU and the partner CPU can use the same connection ID number for the same connection, but the connection ID numbers are not required to match. The connection ID number is relevant only for the PROFINET instructions within the user program of the individual CPU.
- You can use any number for the connection ID of the CPU. However, configuring the connection IDs sequentially from "1" provides an easy method for tracking the number of connections in use for a specific CPU.

Note

Each TSEND_C, TRCV_C or TCON instruction in your user program creates a new connection. It is important to use the correct connection ID for each connection.

The following example shows the communication between two CPUs that utilize 2 separate connections for sending and receiving the data.

- The TSEND_C instruction in CPU_1 links to the TRCV_V in CPU_2 over the first connection ("connection ID 1" on both CPU_1 and CPU_2).
- The TRCV_C instruction in CPU_1 links to the TSEND_C in CPU_2 over the second connection ("connection ID 2" on both CPU_1 and CPU_2).



① TSEND_C on CPU_1 creates a connection and assigns a connection ID to that connection (connection ID 1 for CPU_1).

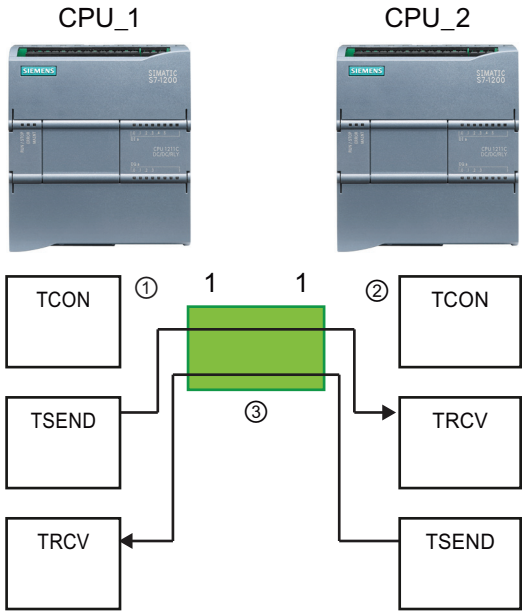
② TRCV_C on CPU_2 creates the connection for CPU_2 and assigns the connection ID (connection ID 1 for CPU_2).

③ TRCV_C on CPU_1 creates a second connection for CPU_1 and assigns a different connection ID for that connection (connection ID 2 for CPU_1).

④ TSEND_C on CPU_2 creates a second connection and assigns a different connection ID for that connection (connection ID 2 for CPU_2).

The following example shows the communication between two CPUs that utilize 1 connection for both sending and receiving the data.

- Each CPU uses a TCON instruction to configure the connection between the two CPUs.
- The TSEND instruction in CPU_1 links to the TRCV instruction in CPU_2 by using the connection ID ("connection ID 1") that was configured by the TCON instruction in CPU_1. The TRCV instruction in CPU_2 links to the TSEND instruction in CPU_1 by using the connection ID ("connection ID 1") that was configured by the TCON instruction in CPU_2.
- The TSEND instruction in CPU_2 links to the TRCV instruction in CPU_1 by using the connection ID ("connection ID 1") that was configured by the TCON instruction in CPU_2. The TRCV instruction in CPU_1 links to the TSEND instruction in CPU_2 by using the connection ID ("connection ID 1") that was configured by the TCON instruction in CPU_1.



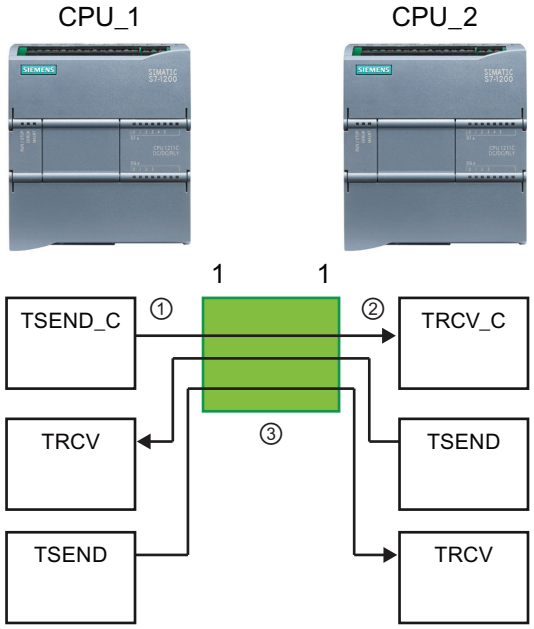
① TCON on CPU_1 creates a connection and assigns a connection ID for that connection on CPU_1 (ID=1).

② TCON on CPU_2 creates a connection and assigns a connection ID for that connection on CPU_2 (ID=1).

③ TSEND and TRCV on CPU_1 use the connection ID created by the TCON on CPU_1 (ID=1).

TSEND and TRCV on CPU_2 use the connection ID created by the TCON on CPU_2 (ID=1).

As shown in the following example, you can also use individual TSEND and TRCV instruction to communication over a connection created by a TSEND_C or TRCV_C instruction. The TSEND and TRCV instructions do not themselves create a new connection, so must use the DB and connection ID that was created by a TSEND_C, TRCV_C or TCON instruction.



① TSEND_C on CPU_1 creates a connection and assigns a connection ID to that connection (ID=1).

② TRCV_C on CPU_2 creates a connection and assigns the connection ID to that connection on CPU_2 (ID=1).

③ TSEND and TRCV on CPU_1 use the connection ID created by the TSEND_C on CPU_1 (ID=1).

TSEND and TRCV on CPU_2 use the connection ID created by the TRCV_C on CPU_2 (ID=1).

10.2.2.2 Protocols

The integrated PROFINET port of the CPU supports multiple communications standards over an Ethernet network:

- Transport Control Protocol (TCP)
- ISO on TCP (RFC 1006)
- User Datagram Protocol (UDP)

Table 10- 1 Protocols and communication instructions for each

Protocol	Usage examples	Entering data in the receive area	Communication instructions	Addressing type
TCP	CPU-to-CPU communication Transport of frames	Ad hoc mode	Only TRCV_C, and TRCV	Assigns port numbers to the Local (active) and Partner (passive) devices
		Data reception with specified length	TSEND_C, TRCV_C, TCON, TDISCON, TSEND, and TRCV	
ISO on TCP	CPU-to-CPU communication Message fragmentation and re-assembly	Ad hoc mode	Only TRCV_C and TRCV	Assigns TSAPs to the Local (active) and Partner (passive) devices
		Protocol-controlled	TSEND_C, TRCV_C, TCON, TDISCON, TSEND, and TRCV	
UDP	CPU-to-CPU communication User program communications	User Datagram Protocol	TUSEND and TURCV	Assigns port numbers to the Local (active) and Partner (passive) devices, but is not a dedicated connection
S7 communication	CPU-to-CPU communication Read/write data from/to a CPU	Data transmission and reception with specified length	GET and PUT	Assigns TSAPs to the Local (active) and Partner (passive) devices
PROFINET RT	CPU-to-PROFINET IO device communication	Data transmission and reception with specified length	Built-in	Built-in

10.2.2.3 Ad hoc mode

Typically, TCP and ISO-on-TCP receive data packets of a specified length, ranging from 1 to 8192 bytes. However, the TRCV_C and TRCV communication instructions also provide an "ad hoc" communications mode that can receive data packets of a variable length from 1 to 1472 bytes.

Note

If you store the data in an "optimized" DB (symbolic only), you can receive data only in arrays of Byte, Char, USInt, and SInt data types.

To configure the TRCV_C or TRCV instruction for ad hoc mode, set the LEN parameter to 65535.

If you do not call the TRCV_C or TRCV instruction in ad hoc mode frequently, you could receive more than one packet in one call. For example: If you were to receive five 100-byte packets with one call, TCP would deliver these five packets as one 500-byte packet, while ISO-on-TCP would restructure the packets into five 100-byte packets.

10.2.2.4 TCP and ISO on TCP

Transport Control Protocol (TCP) is a standard protocol described by RFC 793: Transmission Control Protocol. The primary purpose of TCP is to provide reliable, secure connection service between pairs of processes. This protocol has the following features:

- An efficient communications protocol since it is closely tied to the hardware
- Suitable for medium-sized to large data amounts (up to 8192 bytes)
- Provides considerably more facilities for applications, notably error recovery, flow control, and reliability
- A connection-oriented protocol
- Can be used very flexibly with third-party systems which exclusively support TCP
- Routing-capable
- Only static data lengths are applicable.
- Messages are acknowledged.
- Applications are addressed using port numbers.
- Most of the user application protocols, such as TELNET and FTP, use TCP.
- Programming effort is required for data management due to the SEND / RECEIVE programming interface.

International Standards Organization (ISO) on Transport Control Protocol (TCP) (RFC 1006) (ISO on TCP) is a mechanism that enables ISO applications to be ported to the TCP/IP network. This protocol has the following features:

- An efficient communications protocol closely tied to the hardware
- Suitable for medium-sized to large data amounts (up to 8192 bytes)
- In contrast to TCP, the messages feature an end-of-data identification and are message-oriented.
- Routing-capable; can be used in WAN
- Dynamic data lengths are possible.
- Programming effort is required for data management due to the SEND / RECEIVE programming interface.

Using Transport Service Access Points (TSAPs), TCP protocol allows multiple connections to a single IP address (up to 64K connections). With RFC 1006, TSAPs uniquely identify these communication end point connections to an IP address.

TSEND_C and TRCV_C

The TSEND_C instruction combines the functions of the TCON, TDISCON and TSEND instructions. The TRCV_C instruction combines the functions of the TCON, TDISCON, and TRCV instructions. (Refer to "TCON, TDISCON, TSEND, AND TRCV (Page 408)" for more information on these instructions.)

The minimum size of data that you can transmit (TSEND_C) or receive (TRCV_C) is one byte; the maximum size is 8192 bytes. TSEND_C does not support the transmission of data from boolean locations, and TRCV_C will not receive data into boolean locations. For information transferring data with these instructions, see the section on data consistency (Page 143).

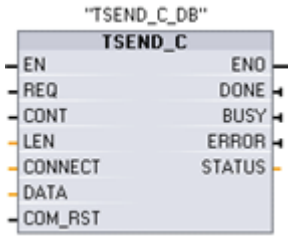
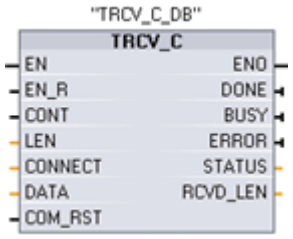
Note

Initializing the communication parameters

After you insert the TSEND_C or TRCV_C instruction, use the "Properties" of the instruction (Page 118) to configure the communication parameters. As you enter the parameters for the communication partners in the inspector window, STEP 7 enters the corresponding data in the DB for the instruction.

If you want to use a multi-instance DB, you must manually configure the DB on both CPUs.

Table 10- 2 TSEND_C and TRCV_C instructions

LAD / FBD	SCL	Description
	"TSEND_C_DB"(req:=_bool_in_, cont:=_bool_in_, len:=_uint_in_, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_, connect:=_struct_inout_, data:=_variant_inout_, com_rst:=_bool_inout_);	TSEND_C establishes a TCP or ISO on TCP communication connection to a partner station, sends data, and can terminate the connection. After the connection is set up and established, it is automatically maintained and monitored by the CPU.
	"TRCV_C_DB"(en_r:=_bool_in_, cont:=_bool_in_, len:=_uint_in_, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_, rcvd_len=>_uint_out_, connect:=_struct_inout_, data:=_variant_inout_, com_rst:=_bool_inout_);	TRCV_C establishes a TCP or ISO on TCP communication connection to a partner CPU, receives data, and can terminate the connection. After the connection is set up and established, it is automatically maintained and monitored by the CPU.

[1] STEP 7 automatically creates the DB when you insert the instruction.

Table 10- 3 TSEND_C and TRCV_C data types for the parameters

Parameter and type		Data type	Description
REQ (TSEND_C)	IN	Bool	Control parameter REQ starts the send job with the connection described in CONNECT on a rising edge.
EN_R (TRCV_C)	IN	Bool	Control parameter enabled to receive: When EN_R = 1, TRCV_C is ready to receive. The receive job is processed.
CONT	IN	Bool	• 0: Disconnect • 1: Establish and hold connection
LEN	IN	UInt	Maximum number of bytes to be sent (TSEND_C) or received (TRCV_C): • Default = 0: The DATA parameter determines the length of the data to be sent (TSEND_C) or received (TRCV_C). • Ad hoc mode = 65535: A variable length of data is set for reception (TRCV_C).
CONNECT	IN_OUT	TCON_Param	Pointer to the connection description
DATA	IN_OUT	Variant	• Contains address and length of data to be sent (TSEND_C) • Contains start address and maximum length of received data (TRCV_C).
COM_RST	IN_OUT	Bool	Allows restart of the instruction: • 0: Irrelevant • 1: Complete restart of the function block, existing connection will be terminated.
DONE	OUT	Bool	• 0: Job is not yet started or still running. • 1: Job completed without error.
BUSY	OUT	Bool	• 0: Job is completed. • 1: Job is not yet completed. A new job cannot be triggered.
ERROR	OUT	Bool	Status parameters with the following values: • 0: No error • 1: Error occurred during processing. STATUS provides detailed information on the type of error.
STATUS	OUT	Word	Status information including error information. (Refer to the "Error and Status Parameters" table below.)
RCVD_LEN (TRCV_C)	OUT	Int	Amount of data actually received, in bytes

Note

The TSEND_C instruction requires a low-to-high transition at the REQ input parameter to start a send job. The BUSY parameter is then set to 1 during processing. Completion of the send job is indicated by either the DONE or ERROR parameters being set to 1 for one scan. During this time, any low-to-high transition at the REQ input parameter is ignored.

Note

The default setting of the LEN parameter (LEN = 0) uses the DATA parameter to determine the length of the data being transmitted. Ensure that the DATA transmitted by the TSEND_C instruction is the same size as the DATA parameter of the TRCV_C instruction.

TSEND_C operations

The following functions describe the operation of the TSEND_C instruction:

- To establish a connection, execute TSEND_C with CONT = 1.
- After successful establishing of the connection, TSEND_C sets the DONE parameter for one cycle.
- To terminate the communication connection, execute TSEND_C with CONT = 0. The connection will be aborted immediately. This also affects the receiving station. The connection will be closed there and data inside the receive buffer could be lost.
- To send data over an established connection, execute TSEND_C with a rising edge on REQ. After a successful send operation, TSEND_C sets the DONE parameter for one cycle.
- To establish a connection and send data, execute TSEND_C with CONT =1 and REQ = 1. After a successful send operation, TSEND_C sets the DONE parameter for one cycle.

TRCV_C operations

The following functions describe the operation of the TRCV_C instruction:

- To establish a connection, execute TRCV_C with parameter CONT = 1.
- To receive data, execute TRCV_C with parameter EN_R = 1. TRCV_C receives the data continuously when parameters EN_R = 1 and CONT = 1.
- To terminate the connection, execute TRCV_C with parameter CONT = 0. The connection will be aborted immediately, and data could be lost.

TRCV_C handles the same receive modes as the TRCV instruction. The following table shows how data is entered in the receive area.

Table 10- 4 Entering the data into the receive area

Protocol variant	Entering the data in the receive area	Parameter "connection_type"	Value of the LEN parameter	Value of the RCVD_LEN parameter (bytes)
TCP	Ad hoc mode	B#16#11	65535	1 to 1472
TCP	Data reception with specified length	B#16#11	0 (recommended) or 1 to 8192, except 65535	1 to 8192
ISO on TCP	Ad hoc mode	B#16#12	65535	1 to 1472
ISO on TCP	Protocol-controlled	B#16#12	0 (recommended) or 1 to 8192, except 65535	1 to 8192

Note

Ad hoc mode

The "ad hoc mode" exists with the TCP and ISO on TCP protocol variants. You set "ad hoc mode" by assigning "65535" to the LEN parameter. The receive area is identical to the area formed by DATA. The length of the received data will be output to the parameter RCVD_LEN.

If you store the data in an "optimized" DB (symbolic only), you can receive data only in arrays of Byte, Char, USInt, and SInt data types.

Note

Importing of S7-300/400 STEP 7 projects containing "ad hoc mode" into the S7-1200

In S7-300/400 STEP 7 projects, "ad hoc mode" is selected by assigning "0" to the LEN parameter. In the S7-1200, you set "ad hoc mode" by assigning "65535" to the LEN parameter.

If you import an S7-300/400 STEP 7 project containing "ad hoc mode" into the S7-1200, you must change the LEN parameter to "65535".

Note

Due to the asynchronous processing of TSEND_C, you must keep the data in the sender area consistent until the DONE parameter or the ERROR parameter assumes the value TRUE.

For TSEND_C, a TRUE state at the parameter DONE means that the data was sent successfully. It does not mean that the connection partner CPU actually read the receive buffer.

Due to the asynchronous processing of TRCV_C, the data in the receiver area are only consistent when parameter DONE = 1.

Table 10- 5 TSEND_C and TRCV_C instructions BUSY, DONE, and ERROR parameters

BUSY	DONE	ERROR	Description
TRUE	irrelevant	irrelevant	The job is being processed.
FALSE	TRUE	FALSE	The job is successfully completed.
FALSE	FALSE	TRUE	The job was ended with an error. The cause of the error can be found in the STATUS parameter.
FALSE	FALSE	FALSE	A new job was not assigned.

Error and Status Parameters

Table 10- 6 TSEND_C and TRCV_C condition codes for ERROR and STATUS

ERROR	STATUS	Description
0	0000	Job executed without error
0	7000	No job processing active
0	7001	Start job processing, establishing connection, waiting for connection partner
0	7002	Data being sent or received
0	7003	Connection being terminated
0	7004	Connection established and monitored, no job processing active
1	8085	LEN parameter is greater than the largest permitted value.
1	8086	The CONNECT parameter is outside the permitted range.
1	8087	Maximum number of connections reached; no additional connection possible.
1	8088	LEN parameter is not valid for the memory area specified in DATA.
1	8089	The CONNECT parameter does not point to a data block.
1	8091	Maximum nesting depth exceeded.
1	809A	The CONNECT parameter points to a field that does not match the length of the connection description.
1	809B	The local_device_id in the connection description does not match the CPU.
1	80A1	Communications error: • The specified connection was not yet established • The specified connection is currently being terminated; transmission over this connection is not possible • The interface is being reinitialized
1	80A3	Attempt being made to terminate a nonexistent connection
1	80A4	IP address of the remote partner connection is invalid. For example, the remote partner IP address is the same as the local partner IP address.
1	80A5	Connection ID is already in use.
1	80A7	Communications error: You called TDISCON before TSEND_C was complete.
1	80B2	The CONNECT parameter points to a data block that was generated with the keyword UNLINKED.
1	80B3	Inconsistent parameters: • Error in the connection description • Local port (parameter local_tsap_id) is already present in another connection description. • ID in the connection description different from the ID specified as parameter

ERROR	STATUS	Description
1	80B4	When using the ISO on TCP (connection_type = B#16#12) to establish a passive connection, condition code 80B4 alerts you that the TSAP entered did not conform to one of the following address requirements: • For a local TSAP length of 2 and a TSAP ID value of either E0 or E1 (hexadecimal) for the first byte, the second byte must be either 00 or 01. • For a local TSAP length of 3 or greater and a TSAP ID value of either E0 or E1 (hexadecimal) for the first byte, the second byte must be either 00 or 01 and all other bytes must be valid ASCII characters. • For a local TSAP length of 3 or greater and the first byte of the TSAP ID does not have a value of either E0 or E1 (hexadecimal), then all bytes of the TSAP ID must be valid ASCII characters. Valid ASCII characters are byte values from 20 to 7E (hexadecimal).
1	80B7	Data type and/or length of the transmitted data does not fit in the area in the partner CPU in which it is to be written.
1	80C3	All connection resources are in use.
1	80C4	Temporary communications error: • The connection cannot be established at this time • The interface is receiving new parameters • The configured connection is currently being removed by a TDISCON.
1	8722	CONNECT parameter: Source area invalid: area does not exist in DB.
1	873A	CONNECT parameter: Access to connection description is not possible (for example, DB not available)
1	877F	CONNECT parameter: Internal error such as an invalid ANY reference
1	893A	Parameter contains the number of a DB that is not loaded.

Connection Ethernet protocols

Every CPU has an integrated PROFINET port, which supports standard PROFINET communications. The TSEND_C and TRCV_C and TSEND and TRCV instructions all support the TCP and ISO on TCP Ethernet protocols.

Refer to "Device Configuration: Configuring the Local/Partner connection path (Page 118)" for more information.

See also

Parameters for the PROFINET connection (Page 120)

TCON, TDISCON, TSEND, AND TRCV

Ethernet communication using TCP and ISO on TCP protocols

Note

TSEND_C and TRCV_C instructions

To help simplify the programming of PROFINET/Ethernet communication, the TSEND_C instruction and the TRCV_C instruction combine the functionality of the TCON, TDISCON. TSEND and TRCV instructions:

- TSEND_C combines the TCON, TDISCON and TSEND instructions.
- TRCV_C combines the TCON, TDISCON and TRCV instructions.

The following instructions control the communication process:

- TCON establishes the TCP/IP connection between the client and server (CPU) PC.
- TSEND and TRCV send and receive data.
- TDISCON breaks the connection.

The minimum size of data that you can transmit (TSEND) or receive (TRCV) is one byte; the maximum size is 8192 bytes. TSEND does not support the transmission of data from boolean locations, and TRCV will not receive data into boolean locations. For information transferring data with these instructions, see the section on data consistency (Page 143).

TCON, TDISCON, TSEND, and TRCV operate asynchronously, which means that the job processing extends over multiple instruction executions. For example, you start a job for setting up and establishing a connection by executing an instruction TCON with parameter REQ = 1. Then you use additional TCON executions to monitor the job progress and test for job completion with parameter DONE.

The following table shows the relationships between BUSY, DONE, and ERROR. Use the table to determine the current job status.

Table 10- 7 Interactions between the BUSY, DONE, and ERROR parameters

BUSY	DONE	ERROR	Description
TRUE	irrelevant	irrelevant	The job is being processed.
FALSE	TRUE	FALSE	The job successfully completed.
FALSE	FALSE	TRUE	The job was ended with an error. The cause of the error can be found in the STATUS parameter.
FALSE	FALSE	FALSE	A new job was not assigned.

TCON and TDISCON

Note

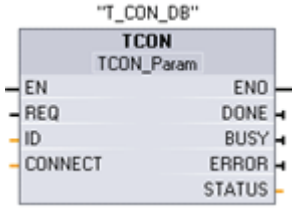
Initializing the communication parameters

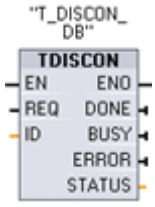
After you insert the TCON instruction, use the "Properties" of the instruction (Page 118) to configure the communication parameters. As you enter the parameters for the communication partners in the inspector window, STEP 7 enters the corresponding data in the instance DB for the instruction.

If you want to use a multi-instance DB, you must manually configure the DB on both CPUs.

Table 10- 8 TCON and TDISCON instructions

LAD / FBD		Description

	`"TCON_DB"(`
	`req:=_bool_in_,`
	`ID:=_undef_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`connect:=_struct_inout_);`
		TCP and ISO on TCP: TCON initiates a communications connection from the CPU to a communication partner.

	`"TDISCON_DB"(`
	`req:=_bool_in_,`
	`ID:=_word_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_);`
		TCP and ISO on TCP: TDISCON terminates a communications connection from the CPU to a communication partner.

[1] STEP 7 automatically creates the DB when you insert the instruction.

Table 10- 9 Data types for the parameters of TCON and TDISCON

Parameter and type		Data type	Description
REQ	IN	Bool	Control parameter REQ starts the job by establishing the connection specified by ID. The job starts at rising edge.
ID	IN	CONN_OUC (Word)	Reference to the connection to be established (TCON) or terminated (TDISCON) to the remote partner, or between the user program and the communication layer of the operating system. The ID must be identical to the associated parameter ID in the local connection description. Value range: W#16#0001 to W#16#0FFF
CONNECT (TCON)	IN_OUT	TCON_Param	Pointer to the connection description

Parameter and type		Data type	Description
DONE	OUT	Bool	• 0: Job is not yet started or still running. • 1: Job completed without error.
BUSY	OUT	Bool	• 0: Job is completed. • 1: Job is not yet completed. A new job cannot be triggered.
ERROR	OUT	Bool	Status parameters with the following values: • 0: No error • 1: Error occurred during processing. STATUS provides detailed information on the type of error.
STATUS	OUT	Word	Status information including error information. (Refer to the Error and Status condition codes in the table below.)

Both communication partners execute the TCON instruction to set up and establish the communication connection. You use parameters to specify the active and passive communication end point partners. After the connection is set up and established, it is automatically maintained and monitored by the CPU.

If the connection is terminated due to a line break or due to the remote communications partner, for example, the active partner attempts to re-establish the configured connection. You do not have to execute TCON again.

An existing connection is terminated and the set-up connection is removed when the TDISCON instruction is executed or when the CPU has gone into STOP mode. To set up and re-establish the connection, you must execute TCON again.

Table 10- 10 ERROR and STATUS condition codes for TCON and TDISCON

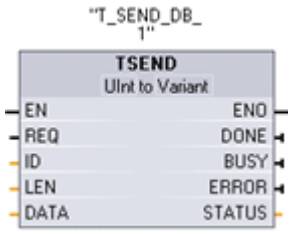
ERROR	STATUS	Description
0	0000	Connection was established successfully.
0	7000	No job processing active
0	7001	Start job processing; establishing connection (TCON) or terminating connection (TDISCON)
0	7002	Follow-on call (REQ irrelevant); establishing connection (TCON) or terminating connection (TDISCON)
1	8086	The ID parameter is outside the permitted address range.
1	8087	TCON: Maximum number of connections reached; no additional connection possible.
1	809B	TCON: The local_device_id in the connection description does not match the CPU.
1	80A1	TCON: Connection or port is already occupied by user.
1	80A2	TCON: Local or remote port is occupied by the system.
1	80A3	Attempt being made to re-establish an existing connection (TCON) or terminate a non-existent connection (TDISCON).
1	80A4	TCON: IP address of the remote connection end point is invalid; it may match the local IP address.
1	80A5	TCON: Connection ID is already in use.
1 ()	80A7	TCON: Communications error: you executed TDISCON before TCON was complete. TDISCON must first completely terminate the connection referenced by the ID.

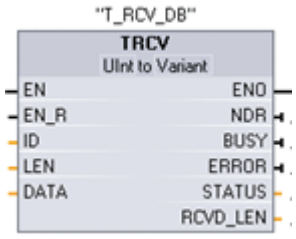
ERROR	STATUS	Description
1	80B4	TCON: When using the ISO on TCP (connection_type = B#16#12) to establish a passive connection, condition code 80B4 alerts you that the TSAP entered did not conform to one of the following address requirements: • For a local TSAP length of 2 and a TSAP ID value of either E0 or E1 (hexadecimal) for the first byte, the second byte must be either 00 or 01. • For a local TSAP length of 3 or greater and a TSAP ID value of either E0 or E1 (hexadecimal) for the first byte, the second byte must be either 00 or 01 and all other bytes must be valid ASCII characters. • For a local TSAP length of 3 or greater and the first byte of the TSAP ID does not have a value of either E0 or E1 (hexadecimal), then all bytes of the TSAP ID must be valid ASCII characters. Valid ASCII characters are byte values from 20 to 7E (hexadecimal).
1	80B6	TCON: Parameter assignment error in parameter connection_type
1	80B7	TCON: Data type and/or length of the transmitted data does not fit in the area in the partner CPU, in which it is to be written.
1)	80B8	TCON: Parameter in the local connection description and Parameter ID are different.
1	80C3	TCON: All connection resources are in use.
1	80C4	Temporary communications error: • The connection cannot be established at this time (TCON). • The configured connection is currently being removed by TDISCON (TCON). • The connection is currently being established (TDISCON). • The interface is receiving new parameters (TCON and TDISCON).

TSEND and TRCV

Table 10- 11 TSEND and TRCV instructions

LAD / FBD	SCL	Description

	`"TSEND_DB"(`
	`req:=_bool_in_,`
	`ID:=_word_in_,`
	`len:=_uint_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`data:=_variant_inout_);`
		TCP and ISO on TCP: TSEND sends data through a communication connection from the CPU to a partner station.

	`"TRCV_DB"(`
	`en_r:=_bool_in_,`
	`ID:=_word_in_,`
	`len:=_uint_in_,`
	`ndr=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`rcvd_len=>_uint_out_,`
	`data:=_variant_inout_);`
		TCP and ISO on TCP: TRCV receives data through a communication connection from a partner station to the CPU.

[1] STEP 7 automatically creates the DB when you insert the instruction.

Table 10- 12 Data types for the parameters of TSEND and TRCV

Parameter and type		Data type	Description
REQ	IN	Bool	TSEND: Starts the send job on a rising edge. The data is transferred from the area specified by DATA and LEN.
EN_R	IN	Bool	TRCV: Enables the CPU to receive; with EN_R = 1, the TRCV is ready to receive. The receive job is processed.
ID	IN	CONN_OUC (Word)	Reference to the associated connection. ID must be identical to the associated parameter ID in the local connection description. Value range: W#16#0001 to W#16#0FFF
LEN	IN	UInt	Maximum number of bytes to be sent (TSEND) or received (TRCV): • Default = 0: The DATA parameter determines the length of the data to be sent (TSEND) or received (TRCV). • Ad hoc mode = 65535: A variable length of data is set for reception (TRCV).
DATA	IN_OUT	Variant	Pointer to send (TSEND) or receive (TRCV) data area; data area contains the address and length. The address refers to I memory, Q memory, M memory, or a DB.

Parameter and type		Data type	Description
DONE	OUT	Bool	TSEND: • 0: Job not yet started or still running. • 1: Job executed without error.
NDR	OUT	Bool	TRCV: • NDR = 0: Job not yet started or still running. • NDR = 1: Job successfully completed.
BUSY	OUT	Bool	• BUSY = 1: The job is not yet complete. A new job cannot be triggered. • BUSY = 0: Job is complete.
ERROR	OUT	Bool	ERROR = 1: Error occurred during processing. STATUS provides detailed information on the type of error
STATUS	OUT	Word	Status information including error information. (Refer to the Error and Status condition codes in the table below.)
RCVD_LEN	OUT	Int	TRCV: Amount of data actually received in bytes

Note

The TSEND instruction requires a low-to-high transition at the REQ input parameter to start a send job. The BUSY parameter is then set to 1 during processing. Completion of the send job is indicated by either the DONE or ERROR parameters being set to 1 for one scan. During this time, any low-to-high transition at the REQ input parameter is ignored.

TRCV Operations

The TRCV instruction writes the received data to a receive area that is specified by the following two variables:

- Pointer to the start of the area
- Length of the area

Note

The default setting of the LEN parameter (LEN = 0) uses the DATA parameter to determine the length of the data being transmitted. Ensure that the DATA transmitted by the TSEND instruction is the same size as the DATA parameter of the TRCV instruction.

As soon as all the job data has been received, TRCV transfers it to the receive area and sets NDR to 1.

Table 10- 13 Entering the data into the receive area

Protocol variant	Entering the data in the receive area	Parameter "connection_type"	Value of the LEN parameter	Value of the RCVD_LEN parameter (bytes)
TCP	Ad hoc mode	B#16#11	65535	1 to 1472
TCP	Data reception with specified length	B#16#11	0 (recommended) or 1 to 8192, except 65535	1 to 8192
ISO on TCP	Ad hoc mode	B#16#12	65535	1 to 1472
ISO on TCP	protocol-controlled	B#16#12	0 (recommended) or 1 to 8192, except 65535	1 to 8192

Note

Ad hoc mode

The "ad hoc mode" exists with the TCP and ISO on TCP protocol variants. You set "ad hoc mode" by assigning "65535" to the LEN parameter. The receive area is identical to the area formed by DATA. The length of the received data will be output to the parameter RCVD_LEN. Immediately after receiving a block of data, TRCV enters the data in the receive area and sets NDR to 1.

If you store the data in an "optimized" DB (symbolic only), you can receive data only in arrays of Byte, Char, USInt, and SInt data types.

Note

Importing of S7-300/400 STEP 7 projects containing "ad hoc mode" into the S7-1200

In S7-300/400 STEP 7 projects, "ad hoc mode" is selected by assigning "0" to the LEN parameter. In the S7-1200, you set "ad hoc mode" by assigning "65535" to the LEN parameter.

If you import an S7-300/400 STEP 7 project containing "ad hoc mode" into the S7-1200, you must change the LEN parameter to "65535".

Table 10- 14 ERROR and STATUS condition codes for TSEND and TRCV

ERROR	STATUS	Description
0	0000	• Send job completed without error (TSEND) • New data accepted: The current length of the received data is shown in RCVD_LEN (TRCV).
0	7000	• No job processing active (TSEND) • Block not ready to receive (TRCV)
0	7001	• Start of job processing, data being sent: During this processing the operating system accesses the data in the DATA send area (TSEND). • Block is ready to receive, receive job was activated (TRCV).

ERROR	STATUS	Description
0	7002	• Follow-on instruction execution (REQ irrelevant), job being processed: The operating system accesses the data in the DATA send area during this processing (TSEND). • Follow-on instruction execution, receive job being processed: Data is written to the receive area during this processing. For this reason, an error could result in inconsistent data in the receive area (TRCV).
1	8085	• LEN parameter is greater than the largest permitted value (TSEND) and (TRCV). • LEN or DATA parameter changed since the first instruction execution (TRCV).
1	8086	The ID parameter is not in the permitted address range.
1	8088	The LEN parameter is larger than the memory area specified in DATA.
1	80A1	Communications error: • The specified connection has not yet established (TSEND and TRCV). • The specified connection is currently being terminated. Transmission or a receive job over this connection is not possible (TSEND and TRCV). • The interface is being reinitialized (TSEND). • The interface is receiving new parameters (TRCV).
1	80C3	Internal lack of resources: A block with this ID is already being processed in a different priority class.
1	80C4	Temporary communications error: • The connection to the communications partner cannot be established at this time. • The interface is receiving new parameter settings, or the connection is currently being established.

Connection Ethernet protocols

Every CPU has an integrated PROFINET port, which supports standard PROFINET communications. The TSEND_C, TRCV_C, TSEND and TRCV instructions all support the TCP and ISO on TCP Ethernet protocols.

Refer to "Device Configuration: Configuring the Local/Partner connection path (Page 118)" for more information.

See also

Parameters for the PROFINET connection (Page 120)

10.2.2.5 UDP

UDP is a standard protocol described by RFC 768: User Datagram Protocol. UDP provides a mechanism for one application to send a datagram to another; however, delivery of data is not guaranteed. This protocol has the following features:

- A quick communications protocol, because it is very hardware-intimate
- Suitable for small-sized to medium data amounts (up to 2048 bytes)

- UDP is a simpler transport control protocol than TCP, with a thin layer that yields low overheads
- Can be used very flexibly with many third-party systems
- Routing-capable
- Uses port numbers to direct the datagrams
- Messages are unacknowledged: The application is required to take responsibility for error recovery and security
- Programming effort is required for data management due to the SEND / RECEIVE programming interface

UDP supports broadcast communication. To use broadcast, you must configure the IP address portion of the ADDR configuration. For example: A CPU with an IP address of 192.168.2.10 and seubnet mask of 255.255.255.0 would use a broadcast address of 192.168.2.255.

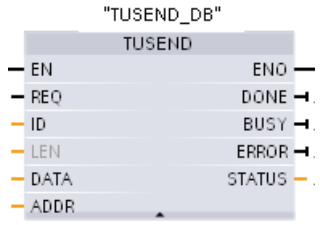
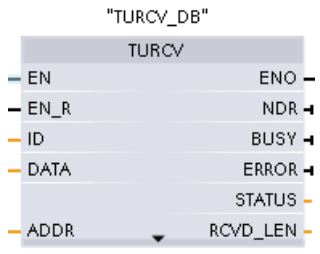
TUSEND and TURCV

The following instructions control the UDP communication process:

- TCON establishes the communication between the client and server (CPU) PC.
- TUSEND and TURCV send and receive data.
- TDISCON disconnects the communication between the client and server.

Refer to TCON, TDISCON, TSEND, and TRCV (Page 408) in the "TCP and ISO-on-TCP" section for more information on the TCON and TDISCON communication instructions.

Table 10- 15 TUSEND and TURCV instructions

LAD / FBD	SCL	Description
	"TUSEND_DB"(req:=_bool_in_, ID:=_word_in_, len:=_uint_in_, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_, data:=_variant_inout_);	The TUSEND instruction sends data via UDP to the remote partner specified by the parameter ADDR. To start the job for sending data, call the TUSEND instruction with REQ = 1.
	"TURCV_DB"(en_r:=_bool_in_, ID:=_word_in_, len:=_uint_in_, ndr=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_, rcvd_len=>_uint_out_, data:=_variant_inout_);	The TURCV instruction receives data via UDP. The parameter ADDR shows the address of the sender. After successful completion of TURCV, the parameter ADDR contains the address of the remote partner (the sender). TURCV does not support ad hoc mode. To start the job for receiving data, call the TURCV instruction with EN_R = 1.

[1] STEP 7 automatically creates the DB when you insert the instruction.

TCON, TDISCON, TUSEND, and TURCV operate asynchronously, which means that the job processing extends over multiple instruction executions.

Table 10- 16 TUSEND and TURCV data types for the parameters

Parameter and type		Data type	Description
REQ (TUSEND)	IN	Bool	Starts the send job on a rising edge. The data is transferred from the area specified by DATA and LEN.
EN_R (TURCV)	IN	Bool	• 0: CPU cannot receive. • 1: Enables the CPU to receive. The TURCV instruction is ready to receive, and the receive job is processed.
ID	IN	Word	Reference to the associated connection between the user program and the communication level of the operating system. ID must be identical to the associated parameter ID in the local connection description. Range of values: W#16#0001 to W#16#0FFF.
LEN	IN	UInt	Number of bytes to be sent (TUSEND) or received (TURCV). • Default = 0. The DATA parameter determines the length of the data to be sent or received. • Otherwise, range of values: 1 to 1472
DONE (TUSEND)	IN	Bool	Status parameter DONE (TUSEND): • 0: Job is not yet started or still running. • 1: Job completed without error.
NDR (TURCV)	OUT	Bool	Status parameter NDR (TURCV): • 0: Job not yet started or still running. • 1: Job has successfully completed.
BUSY	OUT	Bool	• 1: Job is not yet completed. A new job cannot be triggered. • 0: Job has completed.
ERROR	OUT	Bool	Status parameters with the following values: • 0: No error • 1: Error occurred during processing. STATUS provides detailed information on the type of error.
STATUS	OUT	Word	Status information including error information. (Refer to the Error and Status condition codes in the table below.)

Parameter and type		Data type	Description
DATA	IN_OUT	Variant	Address of the sender area (TUSEND) or receive area (TURCV): • The process image input table • The process image output table • A memory bit • A data block
ADDR	IN_OUT	Variant	Pointer to the address of the receiver (for TUSEND) or sender (for TURCV) (for example, P#DB100.DBX0.0 byte 8). The pointer may point to any memory area. A structure of 8 bytes is required as follows: • First 4 bytes contain the remote IP address. • Next 2 bytes specify the remote port number. • Last 2 bytes are reserved.

The job status is indicated at the output parameters BUSY and STATUS. STATUS corresponds to the RET_VAL output parameter of asynchronously functioning instructions.

The following table shows the relationships between BUSY, DONE (TUSEND), NDR (TURCV), and ERROR. Using this table, you can determine the current status of the instruction (TUSEND or TURCV) or when the sending (transmission) / receiving process is complete.

Table 10- 17 Status of BUSY, DONE (TUSEND) / NDR (TURCV), and ERROR parameters

BUSY	DONE / NDR	ERROR	Description
TRUE	irrelevant	irrelevant	The job is being processed.
FALSE	TRUE	FALSE	The job was completed successfully.
FALSE	FALSE	TRUE	The job was ended with an error. The cause of the error can be found in the STATUS parameter..
FALSE	FALSE	FALSE	The instruction was not assigned a (new) job.

[1] Due to the asynchronous function of the instructions: For TUSEND, you must keep the data in the sender area consistent until the DONE parameter or the ERROR parameter assumes the value TRUE. For TURCV, the data in the receiver area are only consistent when the NDR parameter assumes the value TRUE.

Table 10- 18 TUSEND and TURCV condition codes for ERROR and STATUS

ERROR	STATUS	Description
0	0000	• Send job completed without error (TUSEND). • New data were accepted. The current length of the received data is shown in RCVD_LEN (TURCV).
0	7000	• No job processing active (TUSEND) • Block not ready to receive (TURCV)

ERROR	STATUS	Description
0	7001	• Start of job processing, data being sent (TUSEND): During this processing, the operating system accesses the data in the DATA send area. • Block is ready to receive, receive job was activated (TURCV).
0	7002	• Follow-on instruction execution (REQ irrelevant), job being processed (TUSEND): During this processing, the operating system accesses the data in the DATA send area. • Follow-on instruction execution, job being processed: During this processing, the TURCV instruction writes data to the receive area. For this reason, an error could result in inconsistent data in the receive area.
1	8085	LEN parameter is greater than the largest permitted value, has the value 0 (TUSEND), or you changed the value of the LEN or DATA parameter since the first instruction execution (TURCV).
1	8086	The ID parameter is not in the permitted address range.
1	8088	• LEN parameter is larger than the memory area (TUSEND) or receive area (TURCV) specified in DATA. • Receive area is too small (TURCV).
1	8089	ADDR parameter does not point to a data block.
1	80A1	Communications error: • The specified connection between user program and communications layer of the operating system has not yet been established. • The specified connection between the user program and the communication layer of the operating system is currently being terminated. Transmission (TUSEND) or a receive job (TURCV) over this connection is not possible. • The interface is being reinitialized.
1	80A4	IP address of the remote connection end point is invalid; it is possible that it matches the local IP address (TUSEND).
1	80B3	• The set protocol variant (connection_type parameter in the connection description) is not UDP. Please use the TSEND or TRCV instruction. • ADDR parameter: Invalid settings for port number (TUSEND)
1	80C3	• A block with this ID is already being processed in a different priority class. • Internal lack of resources
1	80C4	Temporary communications error: • The connection between the user program and the communication level of the operating system cannot be established at this time (TUSEND). • The interface is receiving new parameters (TUSEND). • The connection is currently being reinitiated (TURCV).

Connection Ethernet protocols

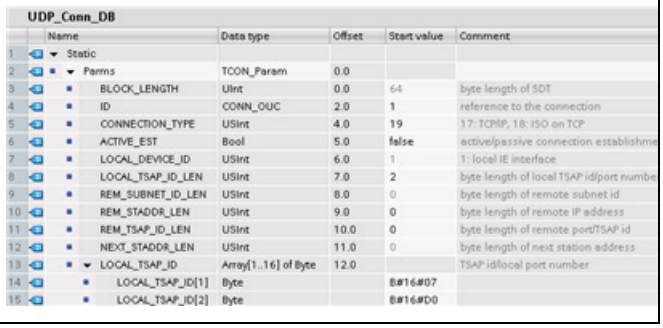
Every CPU has an integrated PROFINET port, which supports standard PROFINET communications. The TUSEND and TURCV instructions support the UDP Ethernet protocol.

Refer to "Configuring the Local/Partner connection path" (Page 118)" in the "Device configuration" chapter for more information.

Operations

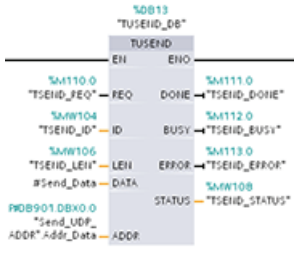
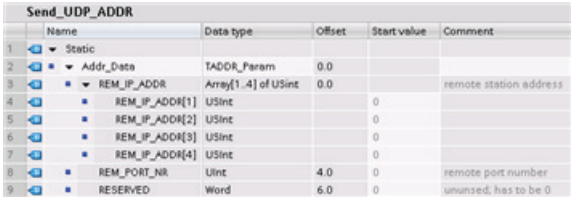
Both partners are passive in UDP communication. Typical parameter start values for the "TCON_Param" data type are shown in the following table. Port numbers (LOCAL_TSAP_ID) are written in a 2-byte format. All ports except for 161, 34962, 34963, and 34964 are allowed.

Table 10- 19 "TCON_Param" data type parameter values

TCON instruction	TCON "UDP Conn DB"
	

The TUSEND instruction sends data through UDP to the remote partner specified in the "TADDR_Param" data type. The TURCV instruction receives data through UDP. After a successful execution of the TURCV instruction, the "TADDR_Param" data type shows the address of the remote partner (the sender).

Table 10- 20 "TADDR_Param" data type parameter values

TUSEND instruction	TUSEND "UDP ADDR DB"
	

10.2.2.6 T_CONFIG

The T_CONFIG instruction changes the IP configuration parameters of the PROFINET port from the user program, allowing the permanent change or setting of the following features:

- Station name
- IP address
- Subnet mask
- Router address

Note

Located in the CPU "Properties", "Ethernet address" page, the "Set IP address using a different method" (Page 426) radio button allows you to change the IP address online or by using the "T_CONFIG" instruction after the program is downloaded. This IP address assignment method is for the CPU only.

Located in the CPU "Properties", "Ethernet address" page, the "Set PROFINET device name using a different method" (Page 427) radio button allows you to change the PROFINET device name online or by using the "T_CONFIG" instruction after the program is downloaded. This PROFINET device name assignment method is for the CPU only.

WARNING

After you use the T_CONFIG to change an IP configuration parameter, the CPU restarts. The CPU will go to STOP mode, warm restart, and return to RUN mode.

Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment.

Ensure that your process will go to a safe state when the CPU restarts as a result of T_CONFIG instruction execution.

Table 10- 21 T_CONFIG instruction

LAD / FBD	SCL	Description

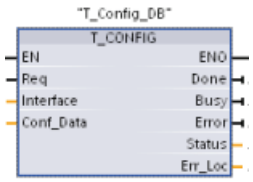
	`"T_CONFIG_DB"(`
	`req:=_bool_in_,`
	`interface:=_word_in_,`
	`conf_Data:=_variant_in_,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_dword_out_,`
	`err_loc=>_word_out_);`
		Use the T_CONFIG instruction to change the IP configuration parameters from your user program.
		T_CONFIG works asynchronously. The execution extends over multiple calls.

Table 10- 22 Data types for the parameters

Parameter and type		Data type	Description
REQ	Input	Bool	Starts the instruction on the rising edge.
INTERFACE	Input	HW_Interface	ID of network interface
CONF_DATA	Input	Variant	Reference to the structure of the configuration data; CONF_DATA is defined by a System Data Type (SDT).
DONE	Output	Bool	• 0: Job has not yet started or is still running. • 1: Job was executed without error.
BUSY	Output	Bool	• 0: The job is complete. • 1: The job is not yet complete. A new job cannot be triggered.
ERROR	Output	Bool	Status parameters with the following values: • 0: No error • 1: Error occurred during processing. STATUS provides detailed information on the type of error.
STATUS	Output	DWord	Status information including error information. (Refer to the Error and Status condition codes in the table below.)
ERR_LOC	Output	DWord	Fault location (field ID and subfield ID of the error parameter)

The IP configuration information is placed in the CONF_DATA data block, along with a Variant pointer on parameter CONF_DATA referenced above. The successful execution of the T_CONFIG instruction ends with the handover of the IP configuration data to the network interface. Errors, such as conflicts between IP addresses, are assigned to the diagnostic buffer and written to the diagnostics buffer.

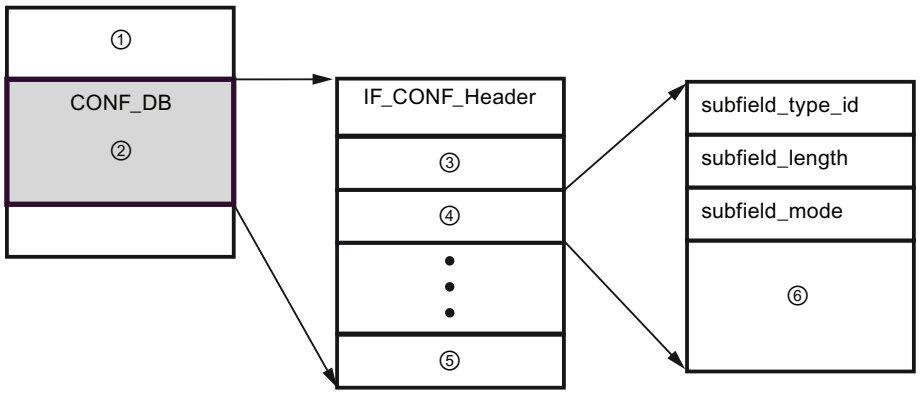
Table 10- 23 Condition codes for ERROR and STATUS

ERROR	STATUS	Description
0	00000000	No error
0	00700000	The job is not finished (BUSY = 1).
0	00700100	Start of job execution
0	00700200	Intermediate call (REQ irrelevant)
1	C08xyy00	General failure
1	C0808000	LADDR parameters for identification of the interface are invalid.
1	C0808100	LADDR parameters for identification of the interface have been assigned a non-supported hardware interface.
1	C0808200	CONF_DATA parameter error: Data type of the Variant pointer does not match the data type Byte.
1	C0808300	CONF_DATA parameter error: The area pointer is not in the DB of the Variant pointer.
1	C0808400	CONF_DATA parameter error: The Variant pointer is the wrong length.
1	C0808600	Reserved
1	C0808700	Inconsistency in the CONF_DATA data block length to the IP configuration
1	C0808800	The parameters of the CONF_DATA data block field_type_id are invalid. (Only field_type_id = 0 is allowed.)

ERROR	STATUS	Description
1	C0808900	The parameters of the CONF_DATA data block field_type_id are invalid or have been used several times.
1	C0808A00	LEN length of the IP configuration parameters or subfield_cnt errors
1	C0808B00	The IP configuration ID parameter is invalid or unsupported.
1	C0808C00	The Sub-block of the IP configuration is incorrectly placed (Sub-block wrong, wrong order, or used multiple times).
1	C0808D00	The length of a statement LEN Sub-blocks is invalid.
1	C0808E00	The value of the parameter in Sub-blocks mode is invalid.
1	C0808F00	Sub-block conflict between the IP configuration and a previous Sub-block.
1	C0809000	The parameters of the subfield are write-protected (for example: parameters are specified by configuration, or PNIO mode is enabled).
1	C0809100	Reserved
1	C0809400	A parameter in the Sub-block IP configuration has not been defined or may not be used.
1	C0809500	There is an inconsistency between a parameter of the Sub-block IP configuration and other parameters.
1	C080C200	Instruction cannot be executed. This error can occur if, for example, communication with the interface has been lost.
1	C080C300	There are not enough resources. This error can occur if, for example, the instruction is called multiple times with different parameters
1	C080C400	Communication failure. The error can occur temporarily and will require a repeat of the user program.
1	C080D200	Execution of the instruction is not supported by the PROFINET interface.

CONF_DATA Data block

The following diagram shows how the configuration data to be transferred is stored in the configuration DB.



① Configuration DB
② Configuration data
③ Subfield 1
④ Subfield 2
⑤ Subfield *n*
⑥ Subfield-specific parameters

The configuration data of CONF_DB consists of a field that contains a header (IF_CONF_Header) and several subfields. IF_CONF_Header provides the following elements:

- field_type_id (data type UInt): Zero
- field_id (data type UInt): Zero
- subfield_cnt (data type UInt): Number of subfields

Each subfield consists of a header (subfield_type_id, subfield_length, subfield_mode) and the subfield-specific parameters. Each subfield must consist of an even number of bytes. The subfield_mode supports a value of 1 (permanent validity of the configuration data)

Note

Only one field (IF_CONF_Header) is currently allowed. Its parameters field_type_id and field_id must have the value zero. Other fields with different values for field_type_id and field_id are subject to future extensions.

In the IF_CONF_Header field, only two subfields, "addr" (IP address) and "nos" (Name of station) are currently allowed.

Table 10- 24 Subfields supported

subfield_type_id	Data type	Explanation
30	IF_CONF_V4	IP parameters: IP address, subnet mask, router address
40	IF_CONF_NOS	PROFINET IO device name (Name of station)

Table 10- 25 Elements of the IF_CONF_V4 data type

Name		Data type	Start value	Description
Id		UInt	30	subfield_type_id
len		UInt	18	subfield_length
mode		UInt	1	subfield_mode (1: permanent)
InterfaceAddress		IP_V4	-	Interface address
ADDR		Array [1..4] of Byte		
	ADDR[1]	Byte	b#16#C8	IP address high byte: 200
	ADDR[2]	Byte	b#16#0C	IP address high byte: 12
	ADDR[3]	Byte	b#16#01	IP address low byte: 1
	ADDR[4]	Byte	b#16#90	IP address low byte: 144
SubnetMask		IP_V4	-	Subnet mask
ADDR		Array [1..4] of Byte		
	ADDR[1]	Byte	b#16#FF	Subnet mask high byte: 255
	ADDR[2]	Byte	b#16#FF	Subnet mask high byte: 255
	ADDR[3]	Byte	b#16#FF	Subnet mask low byte: 255
	ADDR[4]	Byte	b#16#00	Subnet mask low byte: 0

Name		Data type	Start value	Description
DefaultRouter		IP_V4	-	Default router
ADDR		Array [1..4] of Byte		
	ADDR[1]	Byte	b#16#C8	Router high byte: 200
	ADDR[2]	Byte	b#16#0C	Router high byte: 12
	ADDR[3]	Byte	b#16#01	Router low byte: 1
	ADDR[4]	Byte	b#16#01	Router low byte: 1

Table 10- 26 Elements of the IF_CONF_NOS data type

Name	Data type	Start value	Description
id	UInt	40	subfield_type_id
len	UInt	246	subfield_length
mode	UInt	1	subfield_mode (1: permanent)
Nos (Name of station)	Array[1..240] of Byte	0	Station name: You must occupy the ARRAY from the first byte. If the ARRAY is longer than the station name to be assigned, you must enter a zero byte after the actual station name (in conformity with IEC 61158-6-10). Otherwise, nos is rejected and the "T_CONFIG (Page 421)" instruction enters the error code DW#16#C0809400 in STATUS. If you occupy the first byte with zero, the station name is deleted.

The station name is subject to the following limitations:

- Restricted to a total of 240 characters (lower case letters, numbers, dash, or dot)
- A name component within the station name, i.e., a character string between two dots, must not exceed 63 characters.
- No special characters such as umlauts, brackets, underscore, slash, blank space, etc. The only special character permitted is the dash.
- The station name must not begin or end with the "-" character.
- The station name must not begin with a number.
- The station name form n.n.n.n (n = 0, ... 999) is not permitted.
- The station name must not begin with the string "port-xyz" or "port-xyz-abcde" (a, b, c, d, e, x, y, z = 0, ... 9).

Note

You can also create an ARRAY "nos" that is shorter then 240 bytes, but not less than 2 bytes. In this case, you must adjust the "len" (length of subfield) tag accordingly.

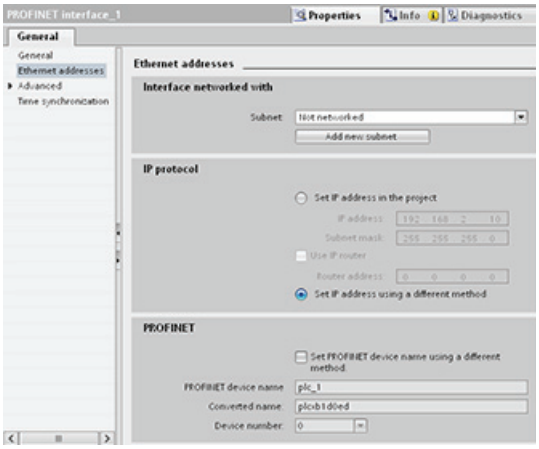
How to change IP parameters

In the following example, in the "addr" subfield, the "InterfaceAddress" (IP address), "SubnetMask", and "DefaultRouter" (IP router) are changed. In the CPU "Properties", "Ethernet address" page, the "Set IP address using a different method" radio button must be clicked to enable you to change the IP address using the "T_CONFIG" instruction after the program is downloaded.

Table 10- 27 How to change IP parameters

CONF_DATA_1

	Name	Data type	Start value
1	Static		
2	Conf_data	Struct	
3	header	IF_CONF_Header	
4	FieldType	UInt	0
5	FieldId	UInt	0
6	SubfieldCount	UInt	1
7	addr	IF_CONF_v4	
8	Id	UInt	30
9	Length	UInt	18
10	Mode	UInt	1
11	InterfaceAddress	IP_V4	
12	ADDR	array [1..4] of Byte	
13	ADDR[1]	Byte	192
14	ADDR[2]	Byte	168
15	ADDR[3]	Byte	2
16	ADDR[4]	Byte	30
17	SubnetMask	IP_V4	
18	ADDR	array [1..4] of Byte	
19	ADDR[1]	Byte	255
20	ADDR[2]	Byte	255
21	ADDR[3]	Byte	255
22	ADDR[4]	Byte	0
23	DefaultRouter	IP_V4	
24	ADDR	array [1..4] of Byte	
25	ADDR[1]	Byte	192
26	ADDR[2]	Byte	168
27	ADDR[3]	Byte	2
28	ADDR[4]	Byte	1



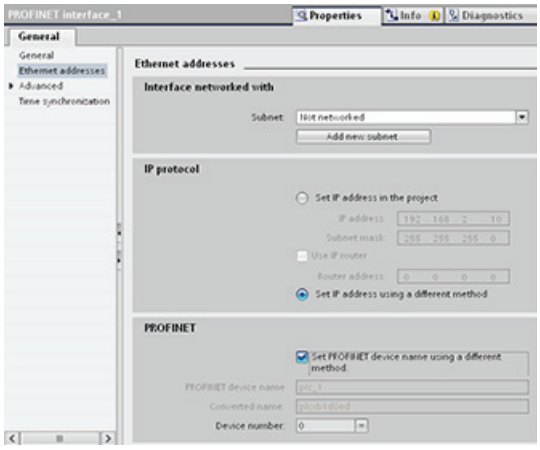
How to change IP parameters and PROFINET IO device names

In the following example, both the "addr" and "nos" (Name of station) subfields are changed. In the CPU "Properties", "Ethernet address" page, the "Set PROFINET device name using a different method" radio button must be clicked to enable you to change the PROFINET device name using the "T_CONFIG" instruction after the program is downloaded.

Table 10- 28 How to change IP parameters and PROFINET IO device names

CONF_DATA_2

	Name	Data type	Start value
1	Static		
2	Conf_data	Struct	
3	header	IF_CONF_Header	
4	FieldType	UInt	0
5	FieldId	UInt	0
6	SubfieldCount	UInt	2
7	addr	IF_CONF_v4	
8	Id	UInt	30
9	Length	UInt	18
10	Mode	UInt	1
11	InterfaceAddress	IP_V4	
12	ADDR	array [1..4] of Byte	
13	SubnetMask	IP_V4	
14	ADDR	array [1..4] of Byte	
15	DefaultRouter	IP_V4	
16	ADDR	array [1..4] of Byte	
17	nos	IF_CONF_NOS	
18	Id	UInt	40
19	Length	UInt	246
20	Mode	UInt	1
21	NOS	array [1..240] of Byte	



10.2.2.7 Common parameters for instructions

REQ input parameter

Many of the Open User Communication instructions use the REQ input to initiate the operation on a low to high transition. The REQ input must be high (TRUE) for one execution of an instruction, but the REQ input can remain TRUE for as long as desired. The instruction does not initiate another operation until it has been executed with the REQ input FALSE so that the instruction can reset the history state of the REQ input. This is required so that the instruction can detect the low to high transition to initiate the next operation.

When you place one of these instructions in your program, STEP 7 prompts you to identify the instance DB. Use a unique DB for each instruction call. This ensures that each instruction properly handles inputs such as REQ.

ID input parameter

This is a reference to the "Local ID (hex)" on the "Network view" of "Devices and networks" in STEP 7 and is the ID of the network that you want to use for this communication block. The ID must be identical to the associated parameter ID in the local connection description.

DONE, NDR, ERROR, and STATUS output parameters

These instructions provide outputs describing the completion status:

Table 10- 29 Open User Communication instruction output parameters

Parameter	Data type	Default	Description
DONE	Bool	FALSE	Is set TRUE for one execution to indicate that the last request completed without errors; otherwise, FALSE.
NDR	Bool	FALSE	Is set TRUE for one execution to indicate that the requested action has completed without error and new data has been received; otherwise, FALSE.
BUSY	Bool	FALSE	Is set TRUE when active to indicate that: • The job is not yet complete. • A new job cannot be triggered. Is set FALSE when job is complete.
ERROR	Bool	FALSE	Is set TRUE for one execution to indicate that the last request completed with errors, with the applicable error code in STATUS; otherwise, FALSE.
STATUS	Word	0	Result status: • If the DONE or NDR bit is set, then STATUS is set to 0 or to an informational code. • If the ERROR bit is set, then STATUS is set to an error code. • If none of the above bits are set, then the instruction returns status results that describe the current state of the function. STATUS retains its value for the duration of the execution of the function.

Note

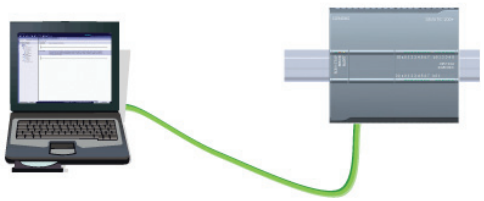
Note that DONE, NDR, and ERROR are set for one execution only.

Restricted TSAPs and port numbers for passive ISO and TCP communication

If you use the "TCON" instruction to set up and establish a passive communications connection, the following port addresses are restricted and should not be used:

- ISO TSAP (passive):
  - 01.00, 01.01, 02.00, 02.01, 03.00, 03.01
  - 10.00, 10.01, 11.00, 11.01, ... BF.00, BF.01
- TCP port (passive): 5001, 102, 123, 20, 21, 25, 34962, 34963, 34964, 80
- UDP port (passive): 161, 34962, 34963, 34964

10.2.3 Communication with a programming device



A CPU can communicate with a STEP 7 programming device on a network.

Consider the following when setting up communications between a CPU and a programming device:

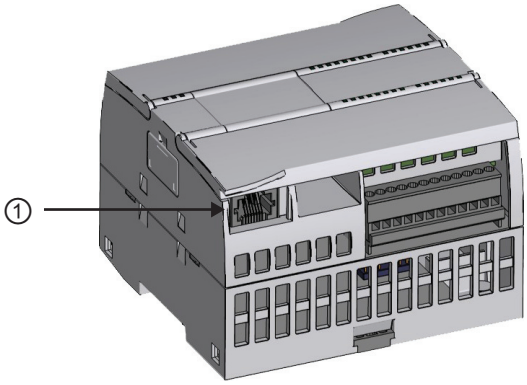
- Configuration/Setup: Hardware configuration is required.
- No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

10.2.3.1 Establishing the hardware communications connection

The PROFINET interfaces establish the physical connections between a programming device and a CPU. Since Auto-Cross-Over functionality is built into the CPU, either a standard or crossover Ethernet cable can be used for the interface. An Ethernet switch is not required to connect a programming device directly to a CPU.

Follow the steps below to create the hardware connection between a programming device and a CPU:

1. Install the CPU (Page 44).
2. Plug the Ethernet cable into the PROFINET port shown below.
3. Connect the Ethernet cable to the programming device.



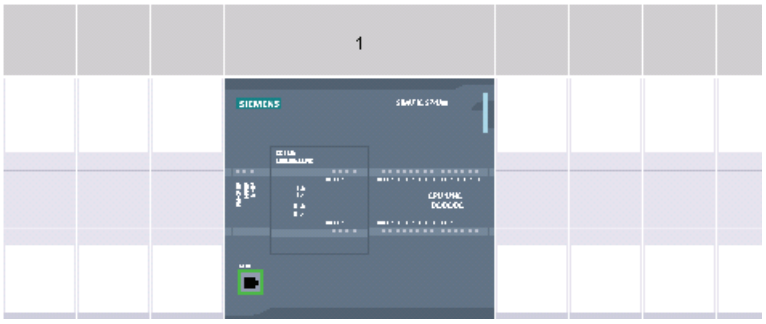
① PROFINET port

An optional strain relief is available to strengthen the PROFINET connection.

10.2.3.2 Configuring the devices

If you have already created a project with a CPU, open your project in the TIA Portal.

If not, create a project and insert a CPU (Page 110) into the rack. In the project below, a CPU is shown in the "Device View".



10.2.3.3 Assigning Internet Protocol (IP) addresses

Assigning the IP addresses

In a PROFINET network, each device must also have an Internet Protocol (IP) address. This address allows the device to deliver data on a more complex, routed network:

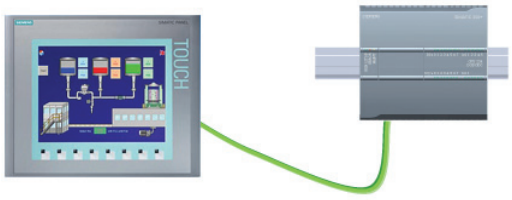
- If you have programming or other network devices that use an on-board adapter card connected to your plant LAN or an Ethernet-to-USB adapter card connected to an isolated network, you must assign IP addresses to them. Refer to "Assigning IP addresses to programming and network devices" (Page 123) for more information.
- You can also assign an IP address to a CPU or network device online. This is particularly useful in an initial device configuration. Refer to "Assigning an IP address to a CPU online" (Page 123) for more information.
- After you have configured your CPU or network device in your project, you can configure parameters for the PROFINET interface, to include its IP address. Refer to "Configuring an IP address for a CPU in your project" (Page 125) for more information.

10.2.3.4 Testing your PROFINET network

After completing the configuration, you must download your project to the CPU. All IP addresses are configured when you download the project.

The CPU "Download to device" function and its "Extended download to device" dialog can show all accessible network devices and whether or not unique IP addresses have been assigned to all devices. Refer to "Testing the PROFINET network" (Page 129) for more information

10.2.4 HMI-to-PLC communication



The CPU supports PROFINET communications connections to HMIs. The following requirements must be considered when setting up communications between CPUs and HMIs:

Configuration/Setup:

- The PROFINET port of the CPU must be configured to connect with the HMI.
- The HMI must be setup and configured.

- The HMI configuration information is part of the CPU project and can be configured and downloaded from within the project.
- No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

Note

The rack-mounted CSM1277 4-port Ethernet switch can be used to connect your CPUs and HMI devices. The PROFINET port on the CPU does not contain an Ethernet switching device.

Supported functions:

- The HMI can read/write data to the CPU.
- Messages can be triggered, based upon information retrieved from the CPU.
- System diagnostics

Table 10- 30 Required steps in configuring communications between an HMI and a CPU

Step	Task
1	Establishing the hardware communications connection A PROFINET interface establishes the physical connection between an HMI and a CPU. Since Auto-Cross-Over functionality is built into the CPU, you can use either a standard or crossover Ethernet cable for the interface. An Ethernet switch is not required to connect an HMI and a CPU. Refer to "Communication with a programming device: Establishing the hardware communications connection" (Page 429) for more information.
2	Configuring the devices Refer to "Communication with a programming device: Configuring the devices" (Page 430) for more information.
3	Configuring the logical network connections between an HMI and a CPU Refer to "HMI-to-PLC communication: Configuring the logical network connections between two devices" (Page 432) for more information.
4	Configuring an IP address in your project Use the same configuration process; however, you must configure IP addresses for the HMI and the CPU. Refer to "Device configuration: Configuring an IP address for a CPU in your project" (Page 127) for more information.
5	Testing the PROFINET network You must download the configuration for each CPU and HMI device. Refer to "Device configuration: Testing the PROFINET network" (Page 129) for more information.

10.2.4.1 Configuring logical network connections between two devices

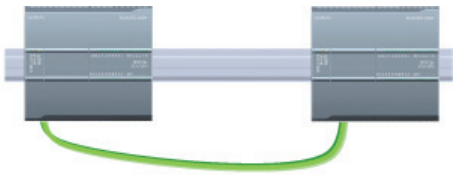
After you configure the rack with the CPU, you are now ready to configure your network connections.

In the Devices and Networks portal, use the "Network view" to create the network connections between the devices in your project. First, click the "Connections" tab, and then select the connection type with the dropdown, just to the right (for example, an ISO on TCP connection).

To create a PROFINET connection, click the green (PROFINET) box on the first device, and drag a line to the PROFINET box on the second device. Release the mouse button and your PROFINET connection is joined.

Refer to "Device Configuration: Creating a network connection" (Page 117) for more information.

10.2.5 PLC-to-PLC communication



A CPU can communicate with another CPU on a network by using the TSEND_C and TRCV_C instructions.

Consider the following when setting up communications between two CPUs:

- Configuration/Setup: Hardware configuration is required.
- Supported functions: Reading/Writing data to a peer CPU
- No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

Table 10- 31 Required steps in configuring communications between two CPUs

Step	Task
1	Establishing the hardware communications connection A PROFINET interface establishes the physical connection between two CPUs. Since Auto-Cross-Over functionality is built into the CPU, you can use either a standard or crossover Ethernet cable for the interface. An Ethernet switch is not required to connect the two CPUs. Refer to "Communication with a programming device: Establishing the hardware communications connection" (Page 429) for more information.
2	Configuring the devices You must configure two CPUs in your project. Refer to "Communication with a programming device: Configuring the devices" (Page 430) for more information.
3	Configuring the logical network connections between two CPUs Refer to "PLC-to-PLC communication: Configuring logical network connections between two devices" (Page 434) for more information.
4	Configuring an IP address in your project Use the same configuration process; however, you must configure IP addresses for two CPUs (for example, PLC_1 and PLC_2). Refer to "Device configuration: Configuring an IP address for a CPU in your project" (Page 127) for more information.

Step	Task
5	Configuring transmit (send) and receive parameters You must configure TSEND_C and TRCV_C instructions in both CPUs to enable communications between them. Refer to "Configuring communications between two CPUs: Configuring transmit (send) and receive parameters" (Page 434) for more information.
6	Testing the PROFINET network You must download the configuration for each CPU. Refer to "Device configuration: Testing the PROFINET network" (Page 129) for more information.

10.2.5.1 Configuring logical network connections between two devices

After you configure the rack with the CPU, you are now ready to configure your network connections.

In the Devices and Networks portal, use the "Network view" to create the network connections between the devices in your project. First, click the "Connections" tab, and then select the connection type with the dropdown, just to the right (for example, an ISO on TCP connection).

To create a PROFINET connection, click the green (PROFINET) box on the first device, and drag a line to the PROFINET box on the second device. Release the mouse button and your PROFINET connection is joined.

Refer to "Device Configuration: Creating a network connection" (Page 117) for more information.

10.2.5.2 Configuring the Local/Partner connection path between two devices

Configuring General parameters

You specify the communication parameters in the "Properties" configuration dialog of the communication instruction. This dialog appears near the bottom of the page whenever you have selected any part of the instruction.

Refer to "Device configuration: Configuring the Local/Partner connection path (Page 118)" for more information.

In the "Address Details" section of the Connection parameters dialog, you define the TSAPs or ports to be used. The TSAP or port of a connection in the CPU is entered in the "Local TSAP" field. The TSAP or port assigned for the connection in your partner CPU is entered under the "Partner TSAP" field.

10.2.5.3 Configuring transmit (send) and receive parameters

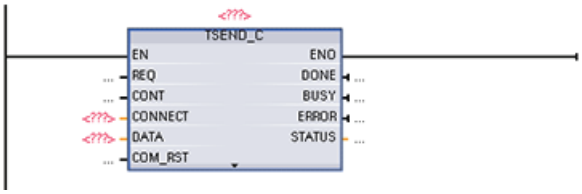
Communication blocks (for example, TSEND_C and TRCV_C) are used to establish connections between two CPUs. Before the CPUs can engage in PROFINET communications, you must configure parameters for transmitting (or sending) messages and receiving messages. These parameters dictate how communications operate when messages are being transmitted to or received from a target device.

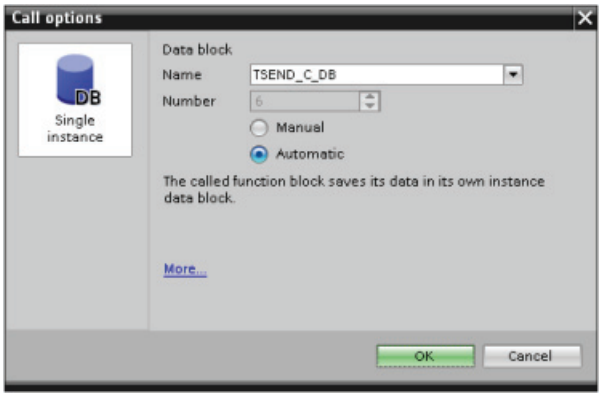
Configuring the TSEND_C instruction transmit (send) parameters

TSEND_C instruction

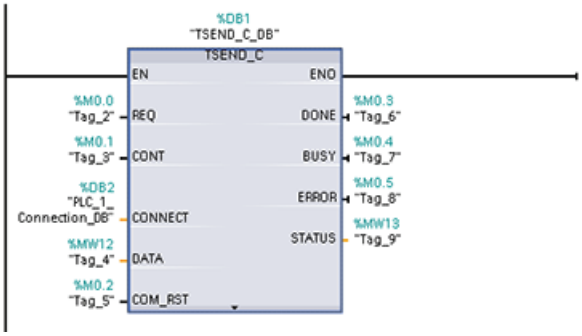
The TSEND_C instruction (Page 401) creates a communications connection to a partner station. The connection is set up, established, and automatically monitored until it is commanded to disconnect by the instruction. The TSEND_C instruction combines the functions of the TCON, TDISCON and TSEND instructions.

From the Device configuration in STEP 7, you can configure how a TSEND_C instruction transmits data. To begin, you insert the instruction into the program from the "Communications" folder in the "Instructions" task card. The TSEND_C instruction is displayed, along with the Call options dialog where you assign a DB for storing the parameters of the instruction.





You can assign tag memory locations to the inputs and outputs, as shown in the following figure:



Configuring General parameters

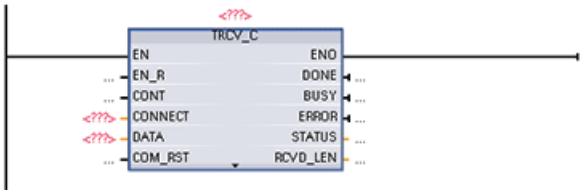
You specify the parameters in the Properties configuration dialog of the TSEND_C instruction. This dialog appears near the bottom of the page whenever you have selected any part of the TSEND_C instruction.

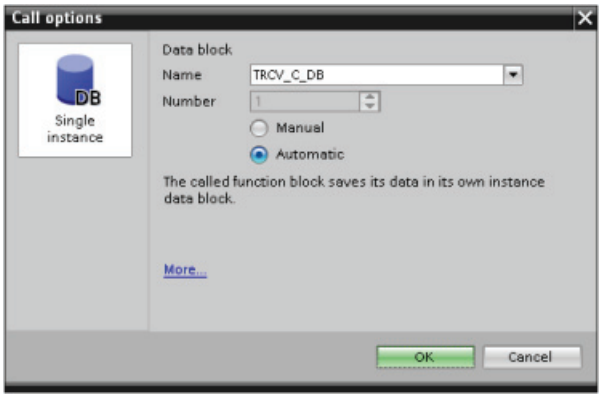
Configuring the TRCV_C instruction receive parameters

TRCV_C instruction

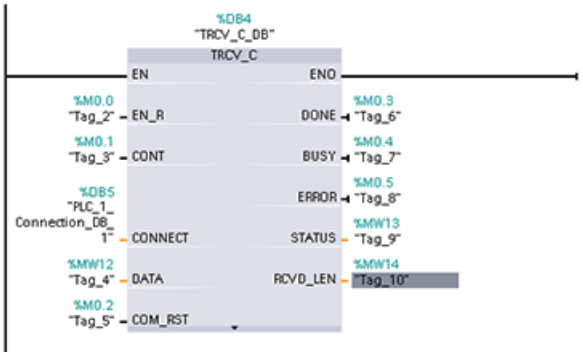
The TRCV_C instruction (Page 401) creates a communications connection to a partner station. The connection is set up, established, and automatically monitored until it is commanded to disconnect by the instruction. The TRCV_C instruction combines the functions of the TCON, TDISCON, and TRCV instructions.

From the CPU configuration in STEP 7, you can configure how a TRCV_C instruction receives data. To begin, insert the instruction into the program from the "Communications" folder in the "Instructions" task card. The TRCV_C instruction is displayed, along with the Call options dialog where you assign a DB for storing the parameters of the instruction.





You can assign tag memory locations to the inputs and outputs, as shown in the following figure:



Configuring the General parameters

You specify the parameters in the Properties configuration dialog of the TRCV_C instruction. This dialog appears near the bottom of the page whenever you have selected any part of the TRCV_C instruction.

10.2.6 Configuring a CPU and PROFINET IO device

Adding a PROFINET IO device

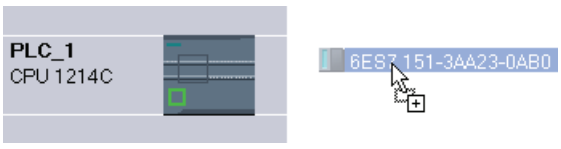
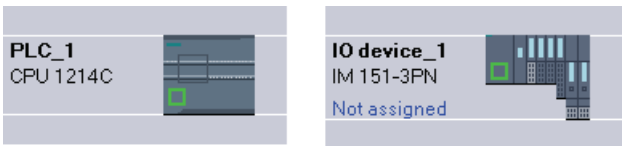
Use the hardware catalog to add PROFINET IO devices.

Note

To add a PROFINET IO device, you can use STEP 7 Professional or Basic, V11 or greater.

For example, expand the following containers in the hardware catalog to add an ET200S IO device: Distributed I/O, ET200S, Interface modules, and PROFINET. You can then select the interface module from the list of ET200S devices (sorted by part number) and add the ET200S IO device.

Table 10- 32 Adding an ET200S IO device to the device configuration

Insert the IO device	Result
	

You can now connect the PROFINET IO device to the CPU:

1. Right-click the "Not assigned" link on the device and select "Assign new IO controller" from the context menu to display the "Select IO controller" dialog.
2. Select your S7-1200 CPU (in this example, "PLC_1") from the list of IO controllers in the project.
3. Click "OK" to create the network connection.

Configuring logical network connections

After you configure the rack with the CPU, you are now ready to configure your network connections.

In the Devices and Networks portal, use the "Network view" to create the network connections between the devices in your project. To create a PROFINET connection, click the green (PROFINET) box on the first device, and drag a line to the PROFINET box on the second device. Release the mouse button and your PROFINET connection is joined.

Refer to "Device Configuration: Creating a network connection" (Page 117) for more information.

Assigning CPUs and device names

Network connections between the devices also assign the PROFINET IO device to the CPU, which is required for that CPU to control the device. To change this assignment, click the PLC Name shown on the PROFINET IO device. A dialog box opens that allows the PROFINET IO device to be disconnected from the current CPU and reassigned or left unassigned, if desired.

The devices on your PROFINET network must have an assigned name before you can connect with the CPU. Use the "Network view" to assign names to your PROFINET devices if the devices have not already been assigned a name or if the name of the device is to be changed. Right-click the PROFINET IO device, and select "Assign device name" to do this.

For each PROFINET IO device, you must assign the same name to that device in both the STEP 7 project and, using the "Online & diagnostics" tool, to the PROFINET IO device configuration memory (for example, an ET200 S interface module configuration memory). If a name is missing or does not match in either location, the PROFINET IO data exchange mode will not run. Refer to "Online and diagnostic tools: Assigning a name to a PROFINET device online (Page 636)" for more information.

Assigning the IP addresses

In a PROFINET network, each device must also have an Internet Protocol (IP) address. This address allows the device to deliver data on a more complex, routed network:

- If you have programming or other network devices that use an on-board adapter card connected to your plant LAN or an Ethernet-to-USB adapter card connected to an isolated network, you must assign IP addresses to them. Refer to "Assigning IP addresses to programming and network devices" (Page 123) for more information.
- You can also assign an IP address to a CPU or network device online. This is particularly useful in an initial device configuration. Refer to "Assigning an IP address to a CPU online" (Page 125) for more information.
- After you have configured your CPU or network device in your project, you can configure parameters for the PROFINET interface, to include its IP address. Refer to "Configuring an IP address for a CPU in your project" (Page 127) for more information.

Configuring the IO cycle time

A PROFINET IO device is supplied with new data from the CPU within an "IO cycle" time period. The update time can be separately configured for each device and determines the time interval in which data is transmitted from the CPU to and from the device.

STEP 7 calculates the "IO cycle" update time automatically in the default setting for each device of the PROFINET network, taking into account the volume of data to be exchanged and the number of devices assigned to this controller. If you do not want to have the update time calculated automatically, you can change this setting.

You specify the "IO cycle" parameters in the "Properties" configuration dialog of the PROFINET IO device. This dialog appears near the bottom of the page whenever you have selected any part of the instruction.

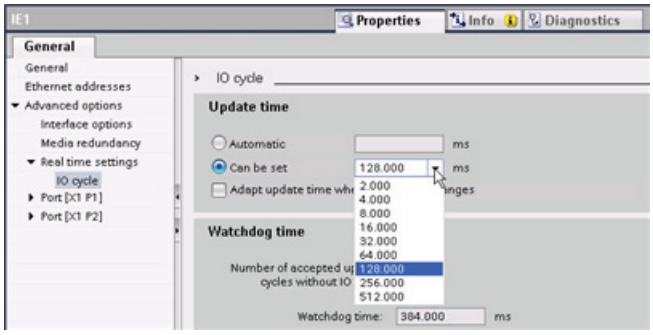
In the "Device view" of the PROFINET IO device, click the PROFINET port. In the "PROFINET Interface" dialog, access the "IO cycle" parameters with the following menu selections:

- "Advanced options"
- "Realtime settings"
- "IO cycle"

Define the IO cycle "Update time" with the following selections:

- To have a suitable update time calculated automatically, select "Automatic".
- To set the update yourself, select "Can be set" and enter the required update time in ms.
- To ensure consistency between the send clock and the update time, activate the "Adapt update time when send clock changes" option. This option ensures that the update time is not set to less than the send clock.

Table 10- 33 Configuring the ET200S PROFINET IO cycle time

ET200 S PROFINET IO device	ET200S PROFINET IO cycle dialog
	

① PROFINET port

10.2.7 Diagnostics

Diagnostic interrupt organization block (OB82)

If a module with diagnostic capability with diagnostic interrupt enabled detects a change in its diagnostic status, it sends a diagnostic interrupt request to the CPU for the following situations:

- A problem has been detected by this module (for example, a wire break) or a component requires maintenance or both (incoming event).
- The problem has been corrected or no longer exists, and no further components require maintenance (outgoing event).

If OB82 does not exist, these errors are written to the diagnostics buffer. The CPU does not take any action or switch to STOP.

If OB82 does exist, the operating system can call OB82 in response to an incoming event. You must create OB82, and this OB allows you to configure local error handling and a more detailed reaction to incoming events.

If you are using a DPV1 capable CPU, you can obtain additional information on the interrupt with the help of the RALRM instruction, which provides more specific information than the start information of OB82.

Peripheral access alarms

These errors are written to the diagnostics buffer. The CPU does not take any action or switch to STOP. Errors written to the diagnostics buffer include:

- Module faults
- Module mismatch
- Module missing

IO access errors

These errors are written to the diagnostics buffer. The CPU does not take any action or switch to STOP.

10.2.8 Distributed I/O Instructions

The following Distributed I/O instructions (Page 260) can be used as indicated with PROFINET:

- RDREC instruction (Page 260): You can read a data record with the number INDEX from a component.
- WRREC instruction (Page 260): You can transfer a data record with the number INDEX to a PROFINET IO device component defined by ID.
- RALRM instruction (Page 263): You can receive an interrupt with all corresponding information from a PROFINET IO device component and supply this information to its output parameters.
- DPRD_DAT instruction (Page 269): The S7-1200 CPU supports up to 64 bytes of consistent data. You must read consistent data areas greater than 64 bytes from a PROFINET IO device with the DPRD_DAT instruction.
- DPWR_DAT instruction (Page 269): The S7-1200 CPU supports up to 64 bytes of consistent data. You must write consistent data areas greater than 64 bytes to a PROFINET IO device with the DPWR_DAT instruction.

10.2.9 Diagnostic instructions

The following diagnostic instructions can be used with either PROFINET or PROFIBUS:

- GET_DIAG instruction (Page 285): You can read the diagnostic information from a specified device.
- DeviceStates instruction (Page 283): You can retrieve the operational states for a distributed I/O device within an I/O subsystem.
- ModuleStates instruction (Page 284): You can retrieve the operational states for the modules in a distributed I/O device.
- LED instruction (Page 282): You can read the state of the LEDs for a distributed I/O device.

10.2.10 Diagnostic events for distributed I/O

Note

With a PROFINET IO system, after a download or power cycle, the CPU will go to RUN mode unless the hardware compatibility is set to allow acceptable substitute modules (Page 114) and one or more modules is missing or is not an acceptable substitute for the configured module.

As shown in the following table, the CPU supports diagnostics that can be configured for the components of the distributed I/O system. Each of these errors generates a log entry in the diagnostic buffer.

Table 10- 34 Handling of diagnostic events for PROFINET and PROFIBUS

Type of error	Diagnostic information for the station?	Entry in the diagnostic buffer?	CPU operating mode
Diagnostic error	Yes	Yes	Stays in RUN mode
Rack or station failure	Yes	Yes	Stays in RUN mode
I/O access error [1]	No	Yes	Stays in RUN mode
Peripheral access error [2]	No	Yes	Stays in RUN mode
Pull / plug event	Yes	Yes	Stays in RUN mode

[1] I/O access error example cause: A module that has been removed.

[2] Peripheral access error example cause: Acyclic communication to a submodule that is not communicating.

Use the GET_DIAG instruction (Page 285) for each station to obtain the diagnostic information. This will allow you to programmatically handle the errors encountered on the device and if desired take the CPU to STOP mode. This method requires you to specify the hardware device from which to read the status information.

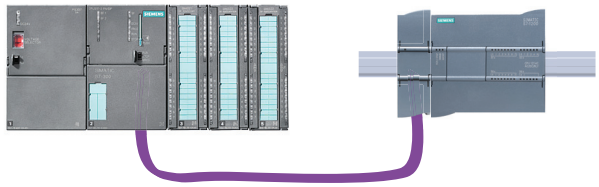
The GET_DIAG instruction uses the "L address" (LADDR) of the station to obtain the health of the entire station. This L Address can be found within the Network Configuration view and by selecting the entire station rack (entire gray area), the L Address is shown in the Properties Tab of the station. You can find the LADDR for each individual module either in the properties for the module (in the device configuration) or in the default tag table for the CPU.

10.3 PROFIBUS

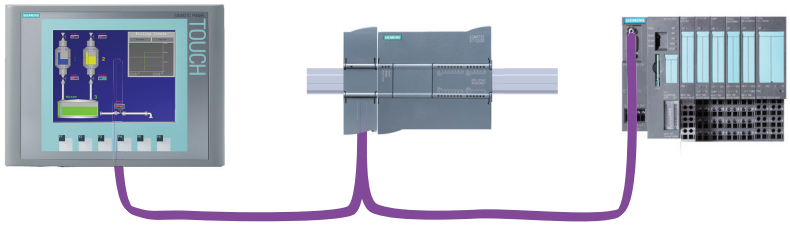
A PROFIBUS system uses a bus master to poll slave devices distributed in a multi-drop fashion on an RS485 serial bus. A PROFIBUS slave is any peripheral device (I/O transducer, valve, motor drive, or other measuring device) which processes information and sends its output to the master. The slave forms a passive station on the network since it does not have bus access rights, and can only acknowledge received messages, or send response messages to the master upon request. All PROFIBUS slaves have the same priority, and all network communication originates from the master.

A PROFIBUS master forms an "active station" on the network. PROFIBUS DP defines two classes of masters. A class 1 master (normally a central programmable controller (PLC) or a PC running special software) handles the normal communication or exchange of data with the slaves assigned to it. A class 2 master (usually a configuration device, such as a laptop or programming console used for commissioning, maintenance, or diagnostics purposes) is a special device primarily used for commissioning slaves and for diagnostic purposes.

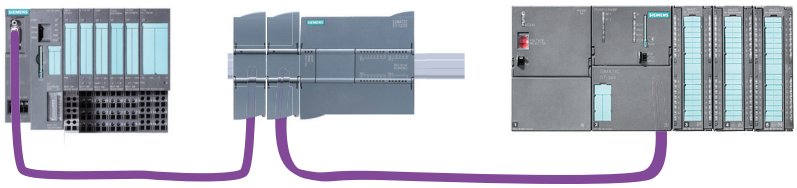
The S7-1200 is connected to a PROFIBUS network as a DP slave with the CM 1242-5 communication module. The CM 1242-5 (DP slave) module can be the communications partner of DP V0/V1 masters. In the figure below, the S7-1200 is a DP slave to an S7-300 controller.



The S7-1200 is connected to a PROFIBUS network as a DP master with the CM 1243-5 communication module. The CM 1243-5 (DP master) module can be the communications partner of DP V0/V1 slaves. In the figure below, the S7-1200 is a master controlling an ET200S DP slave.



If a CM 1242-5 and a CM 1243-5 are installed together, an S7-1200 can perform as both a slave of a higher-level DP master system and a master of a lower-level DP master system, simultaneously.



10.3.1 Communications modules PROFIBUS

10.3.1.1 Connecting to PROFIBUS

Connecting the S7-1200 to PROFIBUS DP

The S7-1200 can be connected to a PROFIBUS fieldbus system with the following communications modules:

- CM 1242-5

  Operates as DP slave

- CM 1243-5

  Operates as DP master class 1

If a CM 1242-5 and a CM 1243-5 are installed together, an S7-1200 can perform the following tasks simultaneously:

- Slave of a higher-level DP master system

  and

- Master of a lower-level DP master system

10.3.1.2 Communications services of the PROFIBUS CMs

Bus protocol

The PROFIBUS CMs use the PROFIBUS DP-V1 protocol.

PROFIBUS communications partners of the S7-1200

The two PROFIBUS CMs allow the S7-1200 to exchange data with the following communications partners.

- CM 1242-5

  The CM 1242-5 (DP slave) can be the communications partner of the following DP V0/V1 masters:

  – SIMATIC S7-1200, S7-300, S7-400, S7 Modular Embedded Controller
  – DP master modules and the distributed IO SIMATIC ET200
  – SIMATIC PC stations
  – SIMATIC NET IE/PB Link
  – Programmable controllers of various vendors

- CM 1243-5

  The CM 1243-5 (DP master) can be the communications partner of the following DP V0/V1 slaves:

  – Distributed I/O SIMATIC ET200
  – S7-1200 CPUs with CM 1242-5
  – S7-200 CPUs with PROFIBUS DP module EM 277
  – SINAMICS converter
  – Drives and actuators from various vendors
  – Sensors of various vendors
  – S7-300/400 CPU with PROFIBUS interface
  – S7-300/400 CPU with PROFIBUS CP (for example CP 342-5)
  – SIMATIC PC stations with PROFIBUS CP

Types of communication with DP-V1

The following types of communication are available with DP-V1:

- Cyclic communication (CM 1242-5 and CM 1243-5)

  Both PROFIBUS modules support cyclic communication for the transfer of process data between DP slave and DP master.

  Cyclic communication is handled by the operating system of the CPU. No software blocks are required for this. The I/O data is read or written directly from/to the process image of the CPU.

- Acyclic communication (CM 1243-5 only)

  The DP master module also supports acyclic communication using software blocks:

  – The "RALRM" instruction is available for interrupt handling.
  – The "RDREC" and "WRREC" instructions are available for transferring configuration and diagnostics data.

Functions not supported by the CM 1243-5:
SYNC/FREEZE
Get_Master_Diag

Other communications services of the CM 1243-5

The CM 1243-5 DP master module supports the following additional communications services:

- S7 communication
  - PUT/GET services

    The DP master functions as a client and server for queries from other S7 controllers or PCs via PROFIBUS.
  - PG/OP communication

    The PG functions allow the downloading of configuration data and user programs from a PG and the transfer of diagnostics data to a PG.

    Possible communications partners for OP communication are HMI panels, SIMATIC panel PCs with WinCC flexible or SCADA systems that support S7 communication.

10.3.1.3 Other properties of the PROFIBUS CMs

Configuration and module replacement

You configure the modules, networks and connections in STEP 7 as of version V11.0.

If you want to configure the module in a third-party system, there is a GSD file available for the CM 1242-5 (DP slave) on the CD that ships with the module and on Siemens Automation Customer Support pages on the Internet.

The configuration data of the PROFIBUS CMs is stored on the local CPU. This allows simple replacement of these communications modules when necessary.

You can configure a maximum of three PROFIBUS CMs per station, of which only one may be a DP master.

Electrical connections

- Power supply
  - The CM 1242-5 is supplied with power via the backplane bus of the SIMATIC station.
  - The CM 1243-5 has a separate connector for the 24 VDC power supply.
- PROFIBUS

  The RS-485 interface of the PROFIBUS connector is a 9-pin D-sub female connector.

  You also have the option of connecting to optical PROFIBUS networks via an Optical Bus Terminal OBT or an Optical Link Module OLM.

Further information

You will find detailed information on the PROFIBUS CMs in the manuals of the devices. You will find these on the Internet on the pages of Siemens Industrial Automation Customer Support under the following entry IDs:

- CM 1242-5:

  49852105 (http://support.automation.siemens.com/WW/view/en/49852105)

- CM 1243-5:

  49851842 (http://support.automation.siemens.com/WW/view/en/49851842)

10.3.1.4 Configuration examples for PROFIBUS

Below, you will find examples of configurations in which the CM 1242-5 is used as a PROFIBUS slave and the CM 1243-5 is used as a PROFIBUS master.

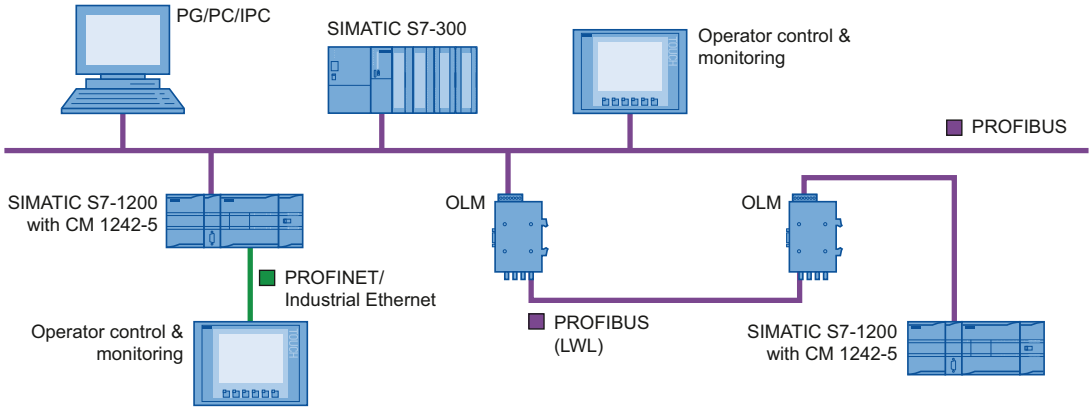


Figure 10-1 Configuration example with a CM 1242-5 as PROFIBUS slave

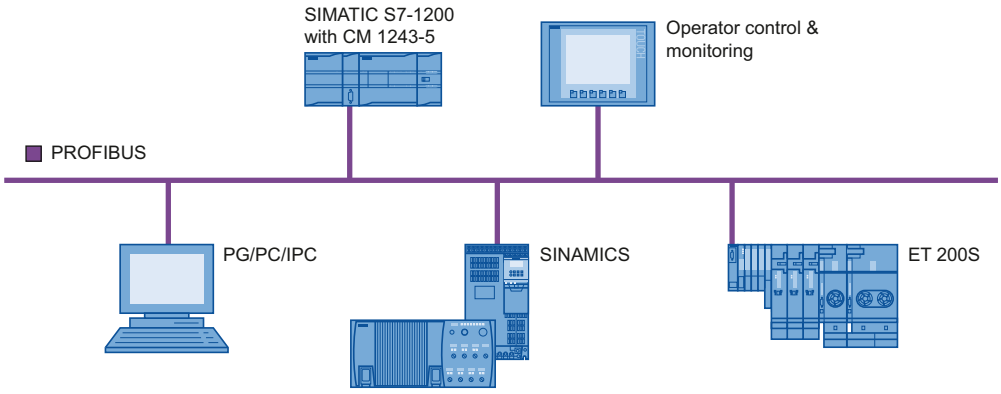


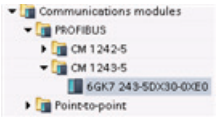
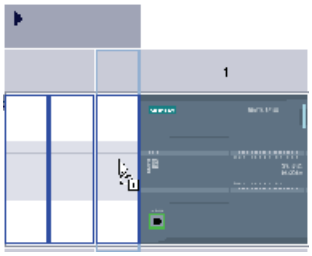
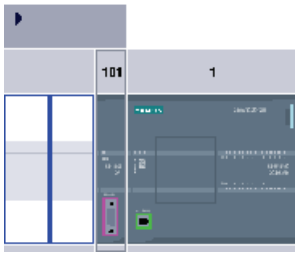
Figure 10-2 Configuration example with a CM 1243-5 as PROFIBUS master

10.3.2 Configuring a DP master and slave device

10.3.2.1 Adding the CM 1243-5 (DP master) module and a DP slave

Use the hardware catalog to add PROFIBUS modules to the CPU. These modules are connected to the left side of the CPU. To insert a module into the hardware configuration, select the module in the hardware catalog and either double-click or drag the module to the highlighted slot.

Table 10- 35 Adding a PROFIBUS CM 1243-5 (DP master) module to the device configuration

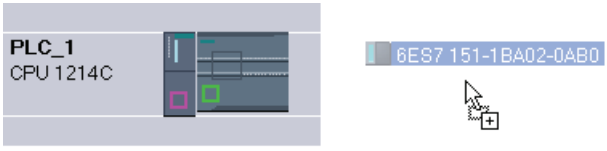
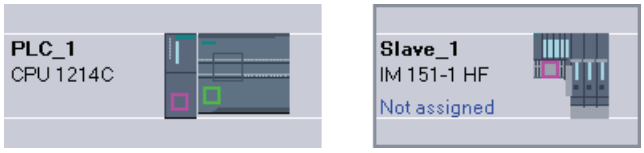
Module	Select the module	Insert the module	Result
CM 1243-5 (DP master)			

Use the hardware catalog to add DP slaves as well. For example, to add an ET200 S DP slave, in the Hardware Catalog, expand the following containers:

- Distributed I/O
- ET200 S
- Interface modules
- PROFIBUS

Next, select "6ES7 151-1BA02-0AB0" (IM151-1 HF) from the list of part numbers, and add the ET200 S DP slave as shown in the figure below.

Table 10- 36 Adding an ET200 S DP slave to the device configuration

Insert the DP slave	Result
	

10.3.2.2 Configuring logical network connections between two PROFIBUS devices

After you configure the CM 1243-5 (DP master) module, you are now ready to configure your network connections.

In the Devices and Networks portal, use the "Network view" to create the network connections between the devices in your project. To create the PROFIBUS connection, select the purple (PROFIBUS) box on the first device. Drag a line to the PROFIBUS box on the second device. Release the mouse button and your PROFIBUS connection is joined.

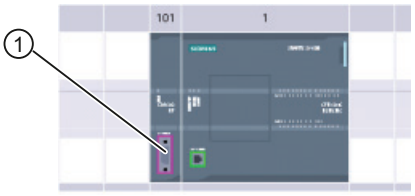
Refer to "Device Configuration: Creating a network connection" (Page 117) for more information.

10.3.2.3 Assigning PROFIBUS addresses to the CM 1243-5 module and DP slave

Configuring the PROFIBUS interface

After you configure logical network connections between two PROFIBUS devices, you can configure parameters for the PROFIBUS interfaces. To do so, click the purple PROFIBUS box on the CM 1243-5 module, and the "Properties" tab in the inspector window displays the PROFIBUS interface. The DP slave PROFIBUS interface is configured in the same manner.

Table 10- 37 Configuring the CM 1243-5 (DP master) module and ET200 S DP slave PROFIBUS interfaces

CM 1243-5 (DP master) module	ET200 S DP slave
	

① PROFIBUS port

Assigning the PROFIBUS address

In a PROFIBUS network, each device is assigned a PROFIBUS address. This address can range from 0 through 127, with the following exceptions:

- Address 0: Reserved for network configuration and/or programming tools attached to the bus
- Address 1: Reserved by Siemens for the first master
- Address 126: Reserved for devices from the factory that do not have a switch setting and must be re-addressed through the network
- Address 127: Reserved for broadcast messages to all devices on the network and may not be assigned to operational devices

Thus, the addresses that may be used for PROFIBUS operational devices are 2 through 125.

In the Properties window, select the "PROFIBUS address" configuration entry. STEP 7 displays the PROFIBUS address configuration dialog, which is used to assign the PROFIBUS address of the device.

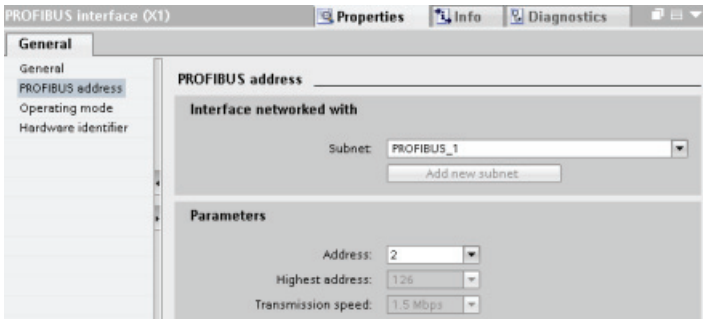


Table 10- 38 Parameters for the PROFIBUS address

Parameter		Description
Subnet	Name of the Subnet to which the device is connected. Click the "Add new subnet" button to create a new subnet. "Not connected" is the default. Two connection types are possible: • The "Not connected" default provides a local connection. • A subnet is required when your network has two or more devices.	
Parameters	Address	Assigned PROFIBUS address for the device
	Highest address	The highest PROFIBUS address is based on the active stations on the PROFIBUS (for example, DP master). Passive DP slaves independently have PROFIBUS addresses from 1 to 125 even if the highest PROFIBUS address is set to 15, for example. The highest PROFIBUS address is relevant for token forwarding (forwarding of the send rights), and the token is only forwarded to active stations. Specifying the highest PROFIBUS address optimizes the bus.
	Transmission rate	Transmission rate of the configured PROFIBUS network: The PROFIBUS transmission rates range from 9.6 Kbits/sec to 12 Mbits/sec. The transmission rate setting depends on the properties of the PROFIBUS nodes being used. The transmission rate should not be greater than the rate supported by the slowest node. The transmission rate is normally set for the master on the PROFIBUS network, with all DP slaves automatically using that same transmission rate (auto-baud).

10.3.3 Distributed I/O Instructions

The following Distributed I/O instructions (Page 260) can be used with PROFIBUS:

- RDREC instruction (Page 260): You can read a data record with the number INDEX from a component.
- WRREC instruction (Page 260): You can transfer a data record with the number INDEX to a DP slave component defined by ID.
- RALRM instruction (Page 263): You can receive an interrupt with all corresponding information from a DP slave component and supply this information to its output parameters.
- DPRD_DAT instruction (Page 269): The CPU supports up to 64 bytes of consistent data. You must read consistent data areas greater than 64 bytes from a DP standard slave with the DPRD_DAT instruction.
- DPWR_DAT instruction (Page 269): The CPU supports up to 64 bytes of consistent data. You must write consistent data areas greater than 64 bytes to a DP standard slave with the DPWR_DAT instruction.
- DPNRM_DG instruction (Page 271): You can read the current diagnostic data of a DP slave in the format specified by EN 50 170 Volume 2, PROFIBUS.

10.3.4 Diagnostic instructions

The following diagnostic instructions can be used with either PROFINET or PROFIBUS:

- GET_DIAG instruction: You can read the diagnostic information from a specified device.
- DeviceStates instruction: You can retrieve the operational states for a distributed I/O device within an I/O subsystem.
- ModuleStates instruction: You can retrieve the operational states for the modules in a distributed I/O device.
- LED instruction: You can read the state of the LEDs for a distributed I/O device.

10.3.5 Diagnostic events for distributed I/O

Note

With a PROFINET IO system, after a download or power cycle, the CPU will go to RUN mode unless the hardware compatibility is set to allow acceptable substitute modules and one or more modules is missing or is not an acceptable substitute for the configured module.

As shown in the following table, the CPU supports diagnostics that can be configured for the components of the distributed I/O system. Each of these errors generates a log entry in the diagnostic buffer.

Table 10- 39 Handling of diagnostic events for PROFINET and PROFIBUS

Type of error	Diagnostic information for the station?	Entry in the diagnostic buffer?	CPU operating mode
Diagnostic error	Yes	Yes	Stays in RUN mode
Rack or station failure	Yes	Yes	Stays in RUN mode
I/O access error [1]	No	Yes	Stays in RUN mode
Peripheral access error [2]	No	Yes	Stays in RUN mode
Pull / plug event	Yes	Yes	Stays in RUN mode

[1] I/O access error example cause: A module that has been removed.

[2] Peripheral access error example cause: Acyclic communication to a submodule that is not communicating.

Use the GET_DIAG instruction for each station to obtain the diagnostic information. This will allow you to programmatically handle the errors encountered on the device and if desired take the CPU to STOP mode. This method requires you to specify the hardware device from which to read the status information.

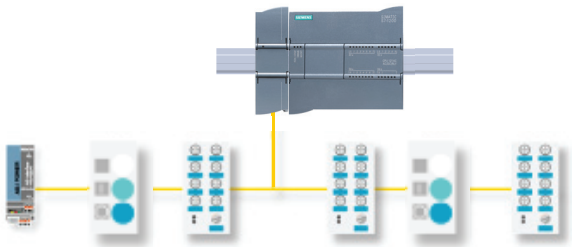
The GET_DIAG instruction uses the "L address" (LADDR) of the station to obtain the health of the entire station. This L Address can be found within the Network Configuration view and by selecting the entire station rack (entire gray area), the L Address is shown in the Properties Tab of the station. You can find the LADDR for each individual module either in the properties for the module (in the device configuration) or in the default tag table for the CPU.

10.4 AS-i

The S7-1200 CM 1243-2 AS-i Master allows the attachment of an AS-i network to an S7-1200 CPU.

The actuator/sensor interface, or AS-i, is a single master network connection system for the lowest level in automation systems. The CM 1243-2 serves as the AS-i master for the network. Using a single AS-i cable, sensors and actuators (AS-i slave devices) can be connected to the CPU through the CM 1243-2. The CM 1243-2 handles all AS-i network coordination and relays data and status information from the actuators and sensors to the CPU through the I/O addresses assigned to the CM 1243-2. You can access binary or analog values depending on the slave type.The AS-i slaves are the input and output channels of the AS-i system and are only active when called by the CM 1243-2.

In the figure below, the S7-1200 is an AS-i master controlling AS-i operator panel and I/O module digital/analog slave devices.

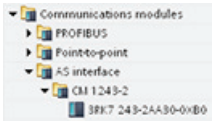
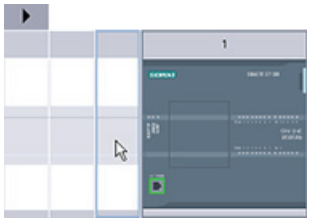
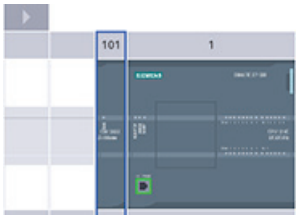


10.4.1 Configuring an AS-i master and slave device

10.4.1.1 Adding the CM 1243-2 AS-i Master module and AS-i slave

Use the hardware catalog to add CM1243-2 AS-i Master modules to the CPU. These modules are connected to the left side of the CPU. To insert a module into the hardware configuration, select the module in the hardware catalog and either double-click or drag the module to the highlighted slot.

Table 10- 40 Adding an AS-i CM 1243-2 AS-i Master module to the device configuration

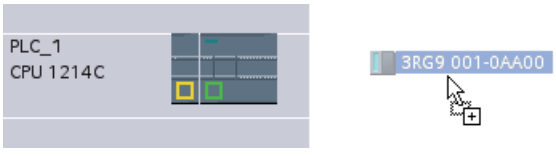
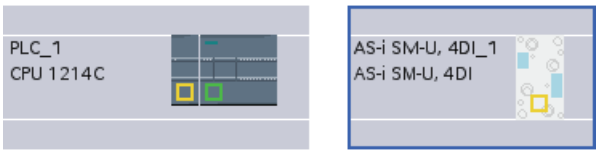
Module	Select the module	Insert the module	Result
CM 1243-2 AS-i Master			

Use the hardware catalog to add AS-i slaves as well. For example, to add an "I/O module, compact, digital, input" slave, in the Hardware Catalog, expand the following containers:

- Field devices
- AS-Interface slaves

Next, select "3RG9 001-0AA00" (AS-i SM-U, 4DI) from the list of part numbers, and add the "I/O module, compact, digital, input" slave as shown in the figure below.

Table 10- 41 Adding an AS-i slave to the device configuration

Insert the AS-i slave	Result
	

10.4.1.2 Configuring logical network connections between two AS-i devices

After you configure the CM 1243-2 AS-i Master module, you are now ready to configure your network connections.

In the Devices and Networks portal, use the "Network view" to create the network connections between the devices in your project. To create the AS-i connection, select the yellow (AS-i) box on the first device. Drag a line to the AS-i box on the second device. Release the mouse button and your AS-i connection is joined.

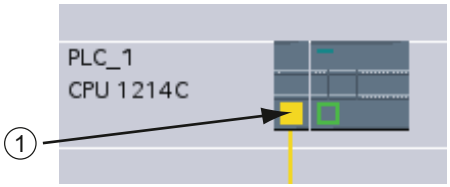
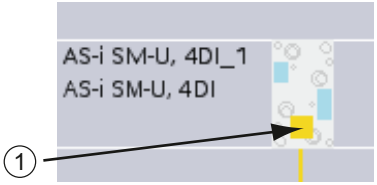
Refer to "Device Configuration: Creating a network connection" (Page 117) for more information.

10.4.1.3 Assigning AS-i addresses to the CM 1243-2 AS-i Master module and AS-i slave

Configuring the AS-i interface

To configure parameters for the AS-i interfaces, click the yellow AS-i box on the CM 1243-2 AS-i Master module, and the "Properties" tab in the inspector window displays the AS-i interface. The AS-i slave interface is configured in the same manner.

Table 10- 42 Configuring the CM 1243-2 AS-i Master module and AS-i I/O module, compact, digital, input slave interface

CM 1243-2 AS-i Master module	AS-i I/O module, compact, digital, input slave
	

① AS-i port

Assigning the AS-i address

In an AS-i network, each device is assigned an AS-i address. This address can range from 0 through 31; however, address 0 is reserved only for new slave devices.

The slave addresses are 1(A or B) to 31(A or B) for a total of up to 62 slave devices. Any address in the range of 1 - 31 can be assigned to an AS-i slave device; in other words, it does not matter whether the slaves begin with address 21 or whether the first slave is actually given the address 1.

A new slave that has not been assigned an address always has address 0. It is detected by the master as a new slave without an address assignment and is not included in normal communication until assigned an address.

In the Properties window, select the "AS-i address" configuration entry. STEP 7 displays the AS-i address configuration dialog, which is used to assign the AS-i address of the device.

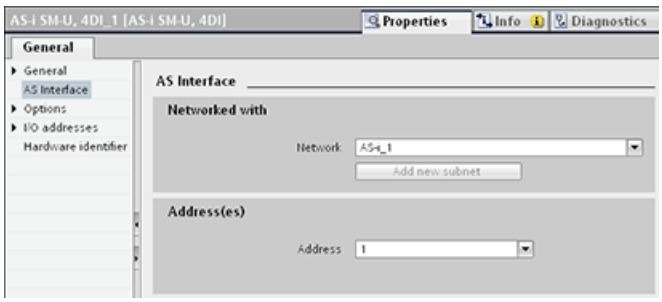


Table 10- 43 Parameters for the AS-i address

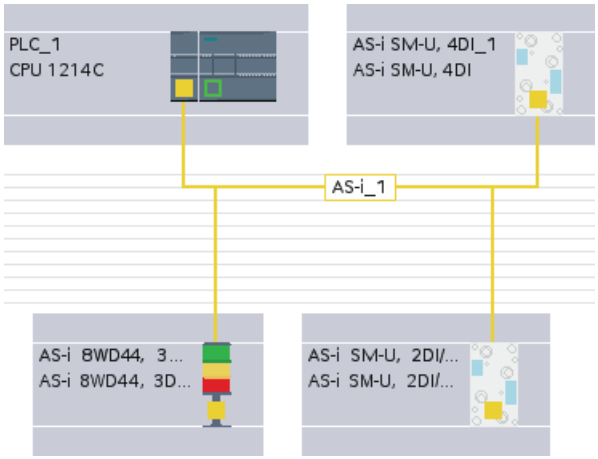
Parameter		Description
Subnet	Name of the Subnet to which the device is connected.	
Parameters	Address	Assigned AS-i address for the slave device in range of 1(A or B) to 31(A or B) for a total of up to 62 slave devices
	Transmission rate	Transmission rate of the configured AS-i network is 10 ms.

10.4.2 Exchanging data between the user program and AS-i slaves

10.4.2.1 Configuring slaves with STEP 7

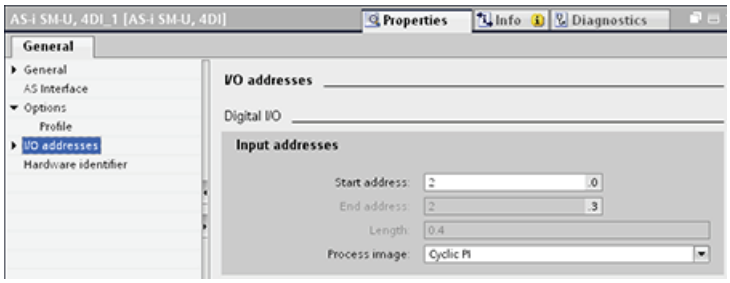
Transferring AS-i digital values

The CPU accesses the digital inputs and outputs of the AS-i slaves through the CM 1243-2 AS-i master in cyclic operation. The data is accessed through I/O addresses or by means of a data record transfer.

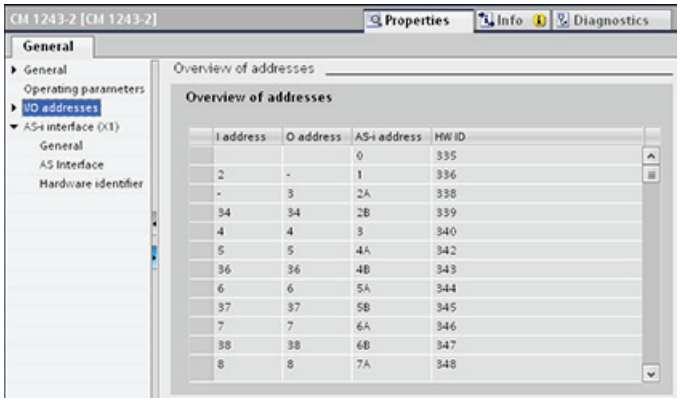


A 62-byte data area is reserved in the I/O area of the SIMATIC. Access to the digital data is performed here in bytes (in other words, one byte is assigned to each AS-i digital slave). When you configure the AS-i slaves in STEP 7, the I/O address for accessing the data from the user program is displayed in the inspection window for the respective AS-i slave.

The digital input module (AS-i SM-U, 4DI) in the AS-i network above has been assigned I/O address 2. By clicking on the digital input module, the Properties for this AS-i slave device are shown below.

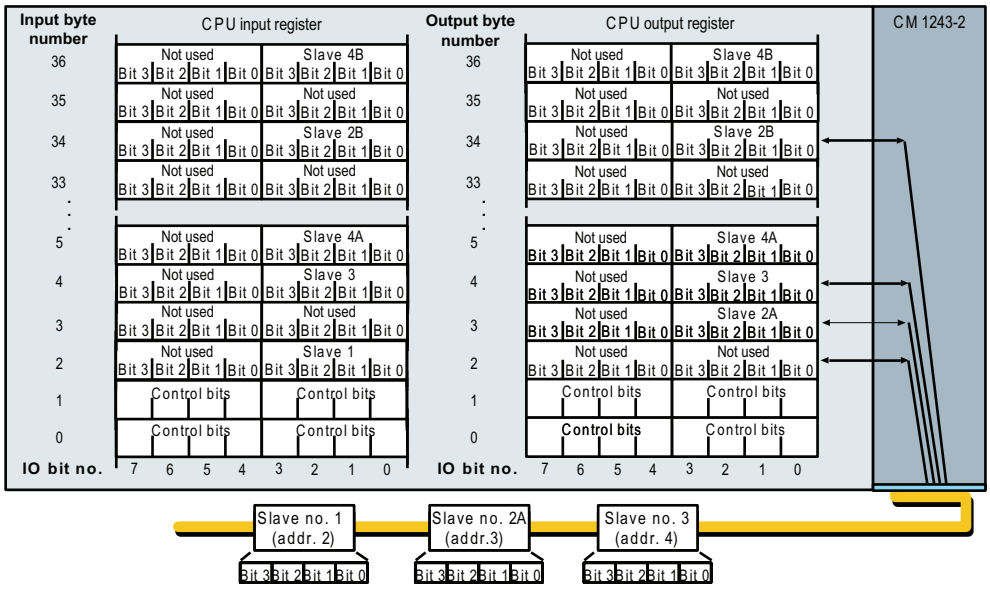


All AS-i slave I/O addresses are shown when viewing the Properties of the CM 1243-2.



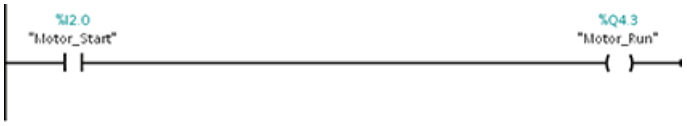
You can access the data of the AS-i slaves in the user program by using the displayed I/O addresses with the appropriate bit logic operations (for example, "AND") or bit assignments.

Bits 0 to 3 are used for user data.



The following simple program illustrates how the assignment works:

Input 2.0 is polled in this program. In the AS-i system, this input belongs to slave1 (Input byte 2, bit 0). Output 4.3, which is then set, corresponds to AS-i slave 3 (Output byte 4, bit 3)



Transferring AS-i analog values

You can access analog data of an AS-i slave through the process image of the PLC if you have configured this AS-i slave in STEP 7 as an analog slave.

If you did not configure the analog slave in STEP 7, you can only access the data of the AS-i slave through the acyclic functions (data record interface). In the user program of the CPU, AS-i calls are read and written using the RDREC (read data record) and WRREC (write data record) distributed I/O instructions.

Note

A configuration of the AS-i slaves specified through STEP 7 and downloaded into the S7 station is transferred by the CPU on the AS-i master CM 1243-2 during S7 station start-up. Any existing configuration that was determined through the "System assignment" online function (Page 457) ("ACTUAL -> EXPECTED") will be overwritten.

Further information

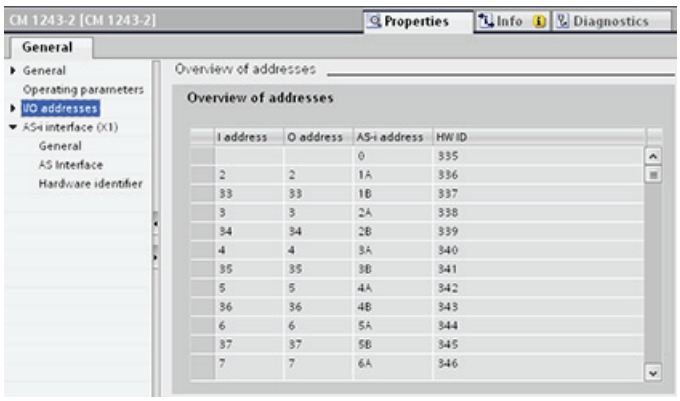
You will find detailed information on the AS-i CM 1243-2 in the "CM 1243-2 and AS-i data decoupling unit DCM 1271 for SIMATIC S7-1200" Manual.

10.4.2.2 Configuring slaves without STEP 7

If the parameter "System assignment" is selected, the system will automatically assign AS-i slave addresses to logical I/O addresses.

This causes the AS-i master to reserve a 62-byte data area in the I/O area of the SIMATIC. Access to the digital data is performed here in bytes (in other words, one byte is assigned to each AS-i digital slave).

The assignment of the AS-i connections of the AS-i digital slaves to the data bits of the assigned byte is indicated in the inspection window of the AS-i master CM 1243-2.



Bits 0 to 3 are used for user data.

You can access the data of the AS-i slaves in the user program by using the displayed I/O addresses with the appropriate bit logic operations (for example, "AND") or bit assignments.

Note

"System assignment" must be activated if you do not configure the AS-i slaves with STEP 7.

If you do not configure any slaves, you must inform the AS-i master about the actual bus configuration using the online function "ACTUAL > EXPECTED" of the TIA Portal.

Further information

You will find detailed information on the AS-i CM 1243-2 in the "CM 1243-2 and AS-i data decoupling unit DCM 1271 for SIMATIC S7-1200" Manual.

10.4.3 Distributed I/O Instructions

The following Distributed I/O instructions (Page 260) can be used as indicated with AS-i:

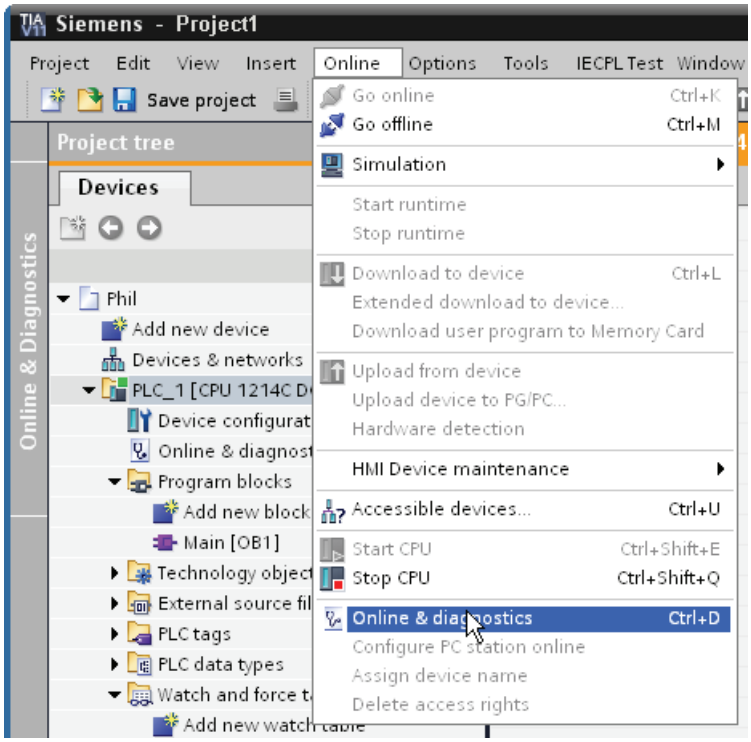
- RDREC instruction (Page 260): You can read a data record with the number INDEX from an AS-i component.
- WRREC instruction (Page 260): You can transfer a data record with the number INDEX to an AS-i component defined by ID.

10.4.4 Working with AS-i online tools

Changing AS-i operational modes online

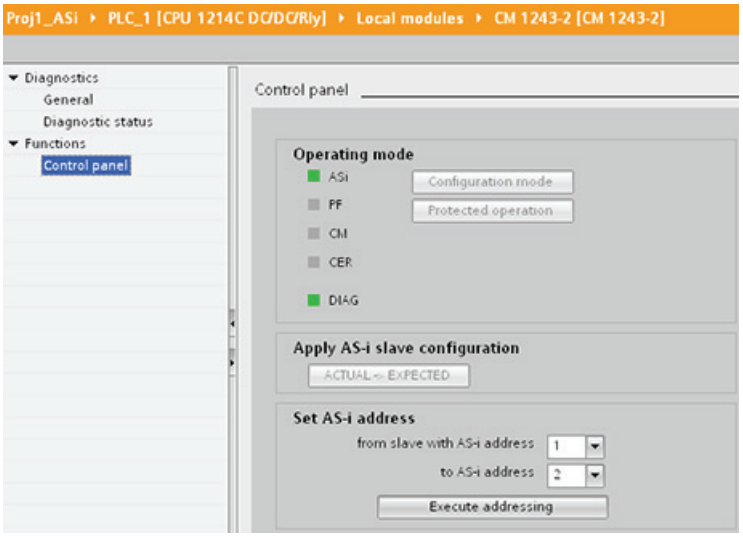
You must go online to view and change the AS-i operational modes.

In order to go online, your must first be in "Device configuration" with the CM1243-2 AS-i Master module selected, and then click the "Go online" button in the toolbar. Next, select the "Online and diagnostics" command from the "Online" menu.



There are two AS-i operational modes:

- Protection mode:
  - You cannot change AS-i slave device and CPU I/O addresses.
  - The green "CM" LED is OFF.
- Configuration mode:
  - You can make required changes in your AS-i slave device and CPU I/O addresses.
  - The green "CM" LED is ON.



Configuration error

When the yellow "CER" LED is ON, there is an error in the AS-i slave device configuration. Select the "ACTUAL > EXPECTED" button to overwrite the CM 1243-2 AS-i master module slave device configuration with the AS-i field network slave device configuration.

10.5 S7 communication

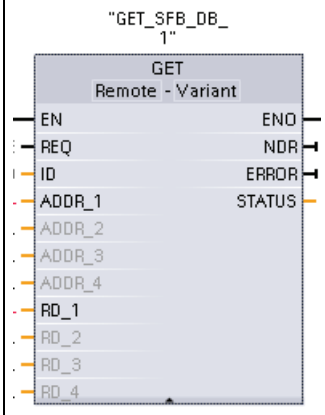
10.5.1 GET and PUT instructions

You can use the GET and PUT instructions to communicate with S7 CPUs through PROFINET and PROFIBUS connections.

- Accessing data in an S7-300/400 CPU: An S7-1200 CPU can use either absolute addresses or symbolic names to address variables of an S7-300/400 CPU. Data types of the remote communication partner that are not supported by the calling S7 1200-CPU may only be accessed as a byte array. For example, S7-300 data type DT is accessed as an array of 8 bytes.
- Accessing data in a standard DB: An S7-1200 CPU can use either absolute addresses or symbolic names to address DB variables in a standard DB of a remote S7 CPU.
- Accessing data in an optimized DB: An S7-1200 CPU can only use symbolic names to address DB variables in an optimized DB of a remote S7 CPU. Only variables of the first nesting level are supported. This includes variables which are declared in an optimized global DB at the DB level. Components of optimized DB structures or elements of arrays cannot be addressed.

Table 10- 44 GET and PUT instructions

LAD / FBD	SCL	Description

	`"GET_DB"(`
	`req:=_bool_in_,`
	`ID:=_word_in_,`
	`ndr=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`addr_1:=_remote_inout_,`
	`[...addr_4:=_remote_inout_,]`
	`rd_1:=_variant_inout_`
	`[,...rd_4:=_variant_inout_]);`
		Use the GET instruction to read data from a remote S7 CPU. The remote CPU can be in either RUN or STOP mode.
		STEP 7 automatically creates the DB when you insert the instruction.

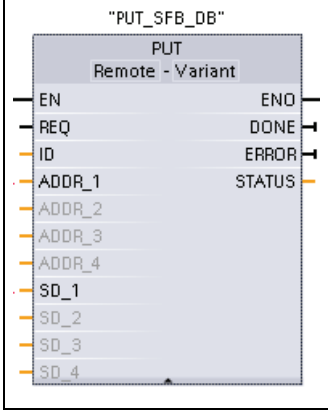
	`"PUT_DB"(`
	`req:=_bool_in_,`
	`ID:=_word_in_,`
	`done=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`addr_1:=_remote_inout_,`
	`[...addr_4:=_remote_inout_,]`
	`sd_1:=_variant_inout_,`
	`[....sd_4:=_variant_inout_]);`
		Use the PUT instruction to write data to a remote S7 CPU. The remote CPU can be in either RUN or STOP mode.
		STEP 7 automatically creates the DB when you insert the instruction.

Table 10- 45 Data types for the parameters

Parameter and type		Data type	Description
REQ	Input	Bool	A low to high (positive edge) signal starts the operation.
ID	Input	CONN_PRG (Word)	Connection identifier
NDR (GET)	Output	Bool	New Data Ready: • 0: request has not yet started or is still running • 1: task was completed successfully
DONE (PUT)	Output	Bool	DONE: • 0: request has not yet started or is still running • 1: task was completed successfully

Parameter and type		Data type	Description
ERROR STATUS	Output Output	Bool Word	• ERROR=0 STATUS value: – 0000H: neither warning nor error – <> 0000H: Warning, STATUS supplies detailed information • ERROR=1 There is an error. STATUS supplies detailed information about the nature of the error.
ADDR_1	InOut	Remote	Pointer to the memory areas in the remote CPU that stores the data to be read (GET) or that is sent (PUT).
ADDR_2	InOut	Remote	
ADDR_3	InOut	Remote	
ADDR_4	InOut	Remote	
RD_1 (GET) SD_1 (PUT)	InOut	Variant	Pointer to the memory areas in the local CPU that stores the data to be read (GET) or sent (PUT). Data types allowed: Bool (only a single bit allowed), Byte, Char, Word, Int, DWord, DInt, or Real. Note: If the pointer accesses a DB, you must specify the absolute address, such as: P# DB10.DBX5.0 Byte 10 In this case, 10 represents the number of bytes to GET or PUT.
RD_2 (GET) SD_2 (PUT)	InOut	Variant	
RD_3 (GET) SD_3 (PUT)	InOut	Variant	
RD_4 (GET) SD_4 (PUT)	InOut	Variant	

You must ensure that the length (number of bytes) and data types for the ADDR_x (remote CPU) and RD_x or SD_x (local CPU) parameters match. The number after the identifier "Byte" is the number of bytes referenced by the ADDR_x, RD_x, or SD_x parameter.

Note

The total number of bytes received on a GET instruction or the total number of bytes sent on a PUT instruction is limited. The limitations are based on how many of the four possible address and memory areas you use:

- If you use only ADDR_1 and RD_1/SD_1, a GET instruction can get 222 bytes and a PUT instruction can send 212 bytes.
- If you use ADDR_1, RD_1/SD_1, ADDR_2, and RD_2/SD_2, a GET instruction can get a total of 218 bytes and a PUT instruction can send a total of 196 bytes.
- If you use ADDR_1, RD_1/SD_1, ADDR_2, RD_2/SD_2, ADDR_3, and RD_3/SD_3 a GET instruction can get a total of 214 bytes and a PUT instruction can send a total of 180 bytes.
- If you use ADDR_1, RD_1/SD_1, ADDR_2, RD_2/SD_2, ADDR_3, RD_3/SD_3, ADDR_4, RD_4/SD_4 a GET instruction can get a total of 210 bytes and a PUT instruction can send a total of 164 bytes.

The sum of the number of bytes of each of your address and memory area parameters must be less than or equal to the defined limits. If you exceed these limits, the GET or PUT instruction returns an error.

On the rising edge of the REQ parameter, the read operation (GET) or write operation (PUT) loads the ID, ADDR_1, and RD_1 (GET) or SD_1 (PUT) parameters.

- For GET: The remote CPU returns the requested data to the receive areas (RD_x), starting with the next scan. When the read operation has completed without error, the NDR parameter is set to 1. A new operation can only be started only after the previous operation has completed.
- For PUT: The local CPU starts sending the data (SD_x) to the memory location (ADDR_x) in the remote CPU. When the write operation has completed without error, the remote CPU returns an execution acknowledgement. The DONE parameter of the PUT instruction is then set to 1. A new write operation can only be started after the previous operation has completed.

Note

To ensure data consistency, always evaluate when the operation has been completed (NDR = 1 for GET, or DONE = 1 for PUT) before accessing the data or initiating another read or write operation.

The ERROR and STATUS parameters provide information about the status of the read (GET) or write (PUT) operation.

Table 10- 46 Error information

ERROR	STATUS (decimal)	Description
0	11	• New job cannot take effect since previous job is not yet completed. • The job is now being processed in a priority class having lower priority.
0	25	Communication has started. The job is being processed.
1	1	Communications problems, such as: • Connection description not loaded (local or remote) • Connection interrupted (for example: cable, CPU is turned off, or CM/CB/CP is in STOP mode) • Connection to partner not yet established
1	2	Negative acknowledgement from the partner device. The task cannot be executed.
1	4	Errors in the send area pointers (RD_x for GET, or SD_x for PUT) involving the data length or the data type.
1	8	Access error on the partner CPU
1	10	Access to the local user memory not possible (for example, attempting to access a deleted DB)
1	12	When the SFB was called: • An instance DB was specified that does not belong to GET or PUT • No instance DB was specified, but rather a shared DB • No instance DB found (loading a new instance DB)

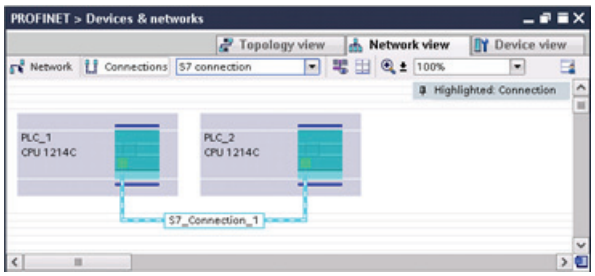
ERROR	STATUS (decimal)	Description
1	20	• Exceeded the maximum number of parallel jobs/instances • The instances were overloaded at CPU-RUN This status is possible for first execution of the GET or PUT instruction
1	27	There is no corresponding GET or PUT instruction in the CPU.

10.5.2 Creating an S7 connection

The connection type that you select creates a communication connection to a partner station. The connection is set up, established, and automatically monitored.

In the Devices and Networks portal, use the "Network view" to create the network connections between the devices in your project. First, click the "Connections" tab, and then select the connection type with the dropdown, just to the right (for example, an S7 connection). Click the green (PROFINET) box on the first device, and drag a line to the PROFINET box on the second device. Release the mouse button and your PROFINET connection is joined.

Refer to "Creating a network connection" (Page 117) for more information.



Click the "Highlighted: Connection" button to access the "Properties" configuration dialog of the communication instruction.

10.5.3 Configuring the Local/Partner connection path between two devices

Configuring General parameters

You specify the communication parameters in the "Properties" configuration dialog of the communication instruction. This dialog appears near the bottom of the page whenever you have selected any part of the instruction.

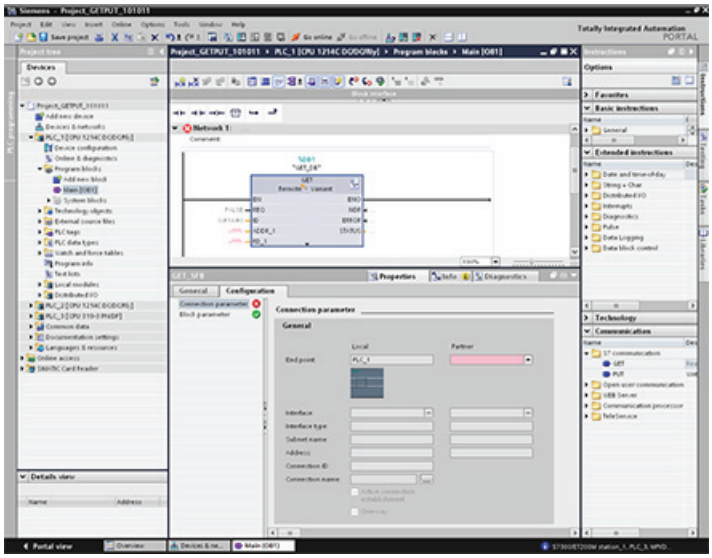
Refer to "Device configuration: Configuring the Local/Partner connection path (Page 118)" for more information.

In the "Address Details" section of the Connection parameters dialog, you define the TSAPs or ports to be used. The TSAP or port of a connection in the CPU is entered in the "Local TSAP" field. The TSAP or port assigned for the connection in your partner CPU is entered under the "Partner TSAP" field.

10.5.4 GET/PUT connection parameter assignment

The GET/PUT instructions connection parameter assignment is a user aid for configuring CPU-to-CPU S7 communication connections.

After inserting a GET or PUT block, the GET/PUT instructions connection parameter assignment is started:



The inspector window displays the properties of the connection whenever you have selected any part of the instruction. You can specify the communication parameters in the "Configuration" tab of the "Properties" for the communication instruction.

10.5.4.1 Connection parameters

The "Connection parameters" page allows the user to configure the necessary S7 connection and to configure the parameter "Connection ID" that is referenced by the GET/PUT block parameter "ID". The page's content has information about the local endpoint and allows the user to define the local interface. The user can also define the partner end point.

The "Block parameters" page allows the user to configure the additional block parameters.

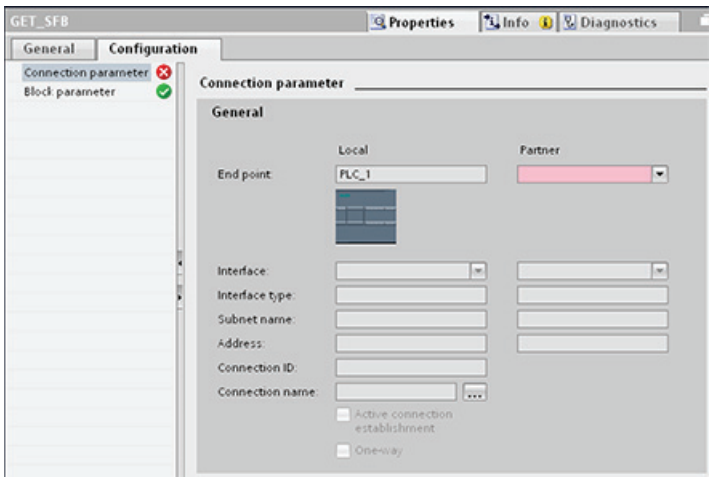


Table 10- 47 Connection parameter: General definitions

Parameter		Definition
Connection parameter: General	End point	"Local End point": Name assigned to the Local CPU "Partner End point": Name assigned to the Partner (remote) CPU Note: In the "Partner End point" dropdown list, the system displays all potential S7 connection partners of the current project as well as the option "unspecified". An unspecified partner represents a communication partner which is not currently in the STEP 7 project (for example, a third party device communication partner).
	Interface	Name assigned to the interfaces Note: The user can change the connection by changing the Local and Partner interfaces
	Interface type	Type of interface
	Subnet name	Name assigned to the subnets
	Address	Assigned IP addresses Note: The user can specify the remote address of a third party device for an "unspecified" communication partner.
	Connection ID	ID number: Automatically generated by the GET/PUT connection parameter assignment
	Connection name	Local and Partner CPU data storage location: Automatically generated by the GET/PUT connection parameter assignment
	Active connection establishment	Checkbox to select Local CPU as the active connection
	One-way	Checkbox to specify a one-way or two-way connection; read-only Note: In a PROFINET GET/PUT connection, both the local and partner devices can act as a server or a client. This allows a two-way connection, and the "One-way" checkbox is unchecked. In a PROFIBUS GET/PUT connection, in some cases, the Partner device can only act as a server (for example, an S7-300), and the "One-way" checkbox is checked.

Connection ID parameter

There are three ways to change the system-defined connection IDs:

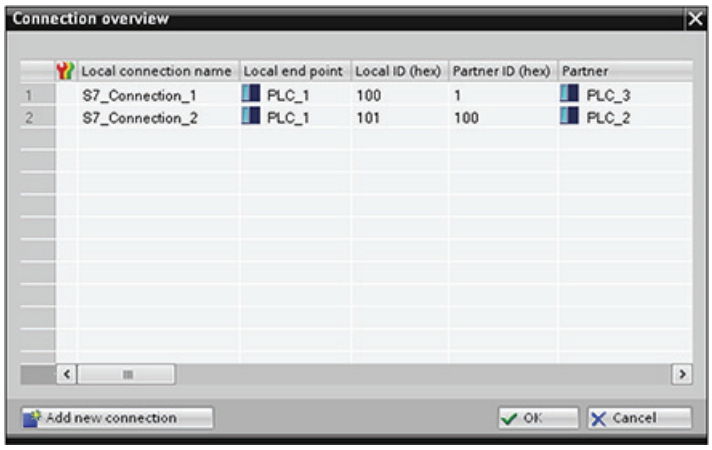
1. The user can change the current ID directly on the GET/PUT block. If the new ID belongs to an already existing connection, the connection is changed.
2. The user can change the current ID directly on the GET/PUT block, but the new ID does not already exist. A new S7 connection is created by the system.
3. The user can change the current ID through the "Connection overview" dialog: The user-input is synchronized with the ID-parameter on the corresponding GET/PUT block.

Note

The parameter "ID" of the GET/PUT block is not a connection name, but a numerical expression which is written like the following example: W#16#1

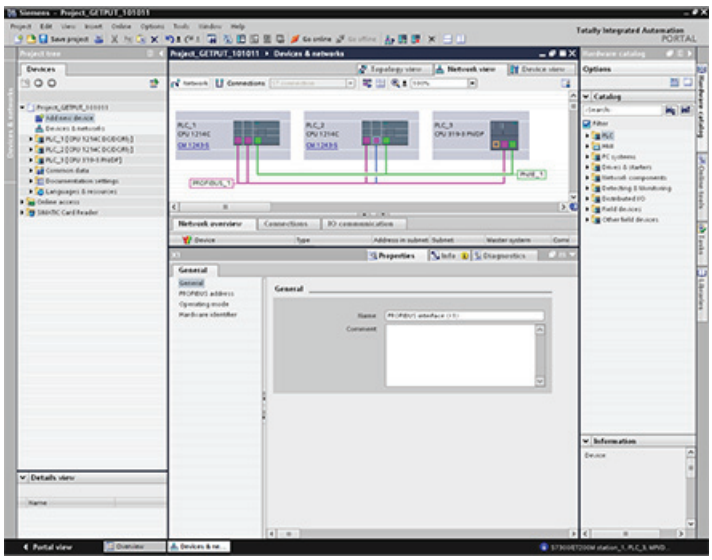
Connection name parameter

The connection name is editable through a special user control, the "Connection overview" dialog. This dialog offers all the available S7 connections which could be selected as an alternative for the current GET/PUT communication. The user can create a completely new connection in this table. Click the button to the right of the "Connection name" field to start the "Connection overview" dialog.



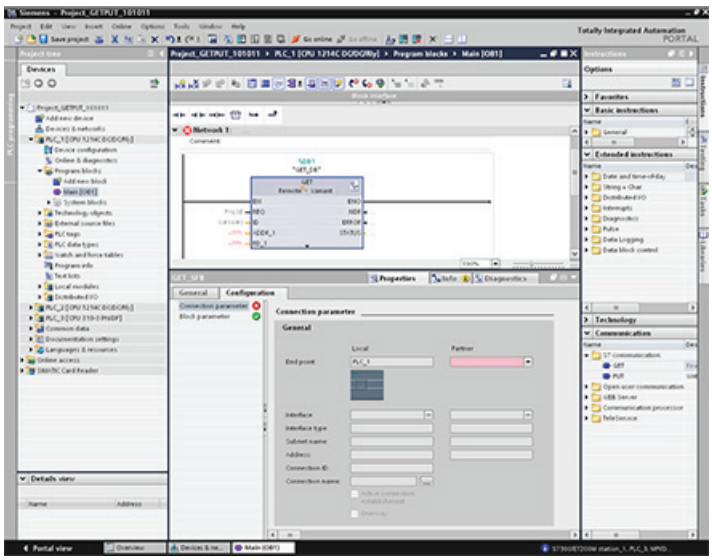
10.5.4.2 Configuring a CPU-to-CPU S7 connection

Given the configuration of PLC_1, PLC_2, and PLC_3 as shown in the figure below, insert GET or PUT blocks for "PLC_1".



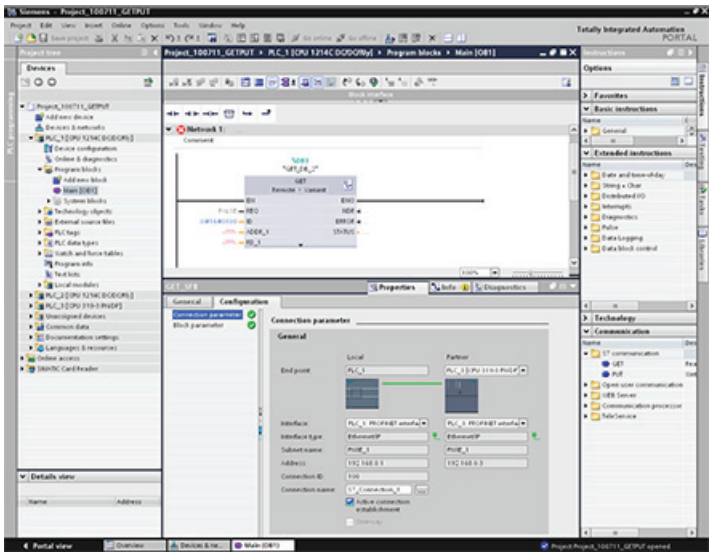
For the GET or PUT instruction, the "Properties" tab is automatically displayed in the inspector window with the following menu selections:

- "Configuration"
- "Connection parameters"



Configuring a PROFINET S7 connection

For the "Partner End point", select "PLC_3".



The system reacts with the following changes:

Table 10- 48 Connection parameter: General values

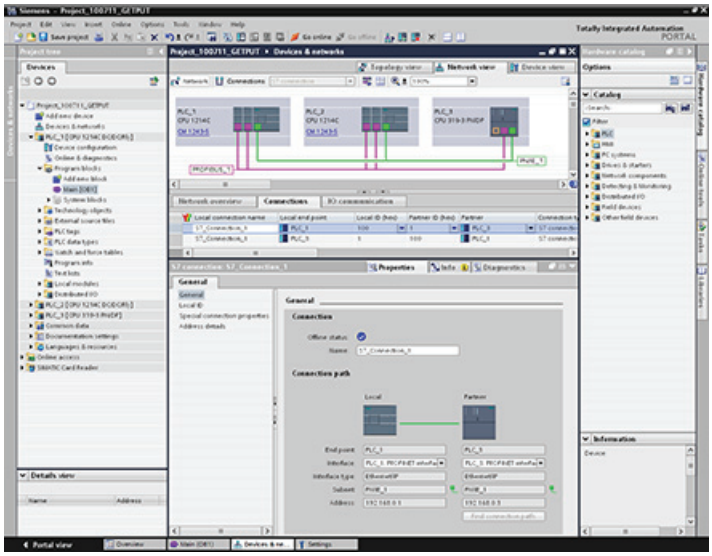
Parameter		Definition
Connection parameter: General	End point	"Local End point" contains "PLC_1" as read-only. "Partner End point" field contains "PLC_3[CPU319-3PN/DP]": • The color switches from red to white • The "Partner" device image is shown. • A connection line appears between the PLC_1- and PLC_3 device images (green Ethernet line).
	Interface	"Local Interface" contains "CPU1214C DC/DC/DC, PROFINET interface (R0/S1)". "Partner Interface" contains: "CPU319-3PN/DP, PROFINET interface (R0/S2)".
	Interface type	"Local Interface type" contains "Ethernet/IP"; control is read-only. "Partner Interface type" contains "Ethernet/IP"; control is read-only. Interface type images are shown at the right beside the Local and Partner "Interface type" (green Ethernet icon).
	Subnet name	"Local Subnet name" contains "PN/IE_1"; control is read only. "Partner Subnet name" contains "PN/IE_1"; control is read only.
	Address	"Local Address" contains the Local IP address; control is read only. "Partner Address" contains the Partner IP address; control is read only.
	Connection ID	"Connection ID" contains "100". In the Program editor, in the Main [OB1], the GET/PUT block "Connection ID" value also contains "100".

Parameter		Definition
	Connection name	"Connection name" contains the default connection name (for example, "S7_Connection_1"); control is enabled.
	Active connection establishment	Checked and enabled to select the Local CPU as the active connection.
	One-way	Read-only and unchecked. Note: "PLC_1" (an S7-1200 CPU 1214CDC/DC/Rly) and "PLC_3" (an S7-300 CPU 319-3PN/DP) can both act as a server and a client in a PROFINET GET/PUT connection, allowing a two-way connection.

The GET/PUT icon in the Property View tree also changes from red to green.

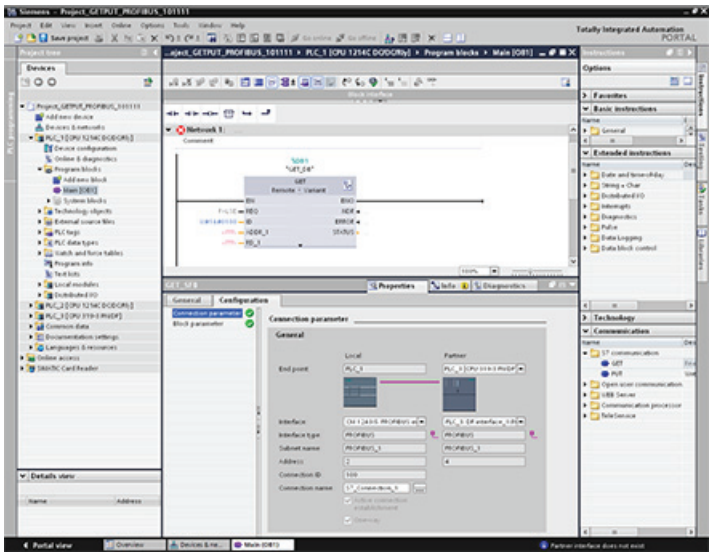
Completed PROFINET S7 connection

In the "Network view", a two-way S7 connection is shown in the "Connections" table between "PLC_1" and "PLC_3".



Configuring a PROFIBUS S7 connection

For the "Partner End point", select "PLC_3".



The system reacts with the following changes:

Table 10- 49 Connection parameter: General values

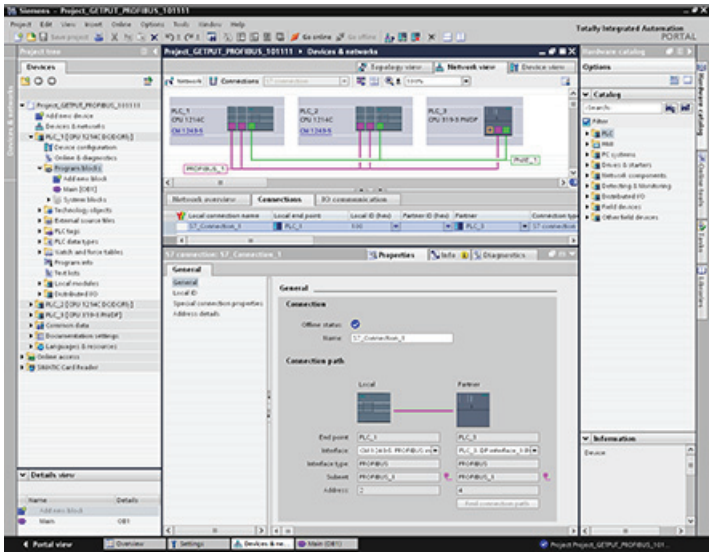
Parameter		Definition
Connection parameter: General	End point	"Local End point" contains "PLC_1" as read-only. "Partner End point" field contains "PLC_3[CPU319-3PN/DP]": • The color switches from red to white • The "Partner" device image is shown. • A connection line appears between the PLC_1- and PLC_3 device images (purple PROFIBUS line).
	Interface	"Local Interface" contains "CPU1214C DC/DC/DC, PROFIBUS interface (R0/S1)". "Partner Interface" contains: "CPU319-3PN/DP, PROFIBUS interface (R0/S2)".
	Interface type	"Local Interface type" contains "PROFIBUS"; control is read-only. "Partner Interface type" contains " PROFIBUS "; control is read-only. Interface type images are shown at the right beside the Local and Partner "Interface type" (purple PROFIBUS icon).
	Subnet name	"Local Subnet name" contains " PROFIBUS _1"; control is read only. "Partner Subnet name" contains " PROFIBUS _1"; control is read only.
	Address	"Local Address" contains the Local IP address; control is read only. "Partner Address" contains the Partner IP address; control is read only.
	Connection ID	"Connection ID" contains "100". In the Program editor, in the Main [OB1], the GET/PUT block "Connection ID" value also contains "100".

Parameter		Definition
	Connection name	"Connection name" contains the default connection name (for example, "S7_Connection_1"); control is enabled.
	Active connection establishment	Read-only, checked, and enabled to select the Local CPU as the active connection.
	One-way	Read-only and checked. Note: "PLC_3" (an S7-300 CPU319-3PN/DP) can act only as a server (cannot also be a client) in a PROFIBUS GET/PUT connection, allowing only a one-way connection.

The GET/PUT icon in the Property View tree also changes from red to green.

Completed PROFIBUS S7 connection

In the "Network view", a one-way S7 connection is shown in the "Connections" table between "PLC_1" and "PLC_3".



Web server

11

The Web server for the S7-1200 provides Web page access to data about your CPU and process data within the CPU.

Standard Web pages

The S7-1200 includes standard Web pages that you can access from your PC from a Web browser (Page 474):

- Introduction (Page 478) - entry point to the standard Web pages
- Start Page (Page 479) - general information about the CPU
- Identification (Page 480) - detailed information about the CPU including serial, order, and version numbers
- Module Information (Page 481) - information about the modules in the local rack
- Communication (Page 483) - information about the network addresses, physical properties of the communication interfaces, and communication statistics
- Diagnostic Buffer (Page 480) - the diagnostic buffer
- Variable Status (Page 484) - CPU variables and I/O, accessible by address or PLC tag name
- Data Logs (Page 486) - data log files stored internally in the CPU or on a memory card

These pages are built in to the S7-1200. For details about the standard Web pages, and how to access them, refer to the Standard web pages (Page 474) section.

User-defined Web pages

The S7-1200 also provides support for you to create user-defined Web pages that can access CPU data. You can develop these pages with the HTML authoring software of your choice, and include pre-defined "AWP" (Automation Web Programming) commands in your HTML code to access CPU data. Refer to the User-defined web pages (Page 492) chapter for specific information on the development of user-defined Web pages, and the associated configuration and programming in STEP 7.

Web browser requirement

The following Web browsers support the Web server:

- Internet Explorer 8.0 or greater
- Mozilla Firefox 3.0 or greater
- Opera 11.0 or greater

For browser-related restrictions that can interfere with the display of standard or user-defined Web pages, see the Constraints (Page 488) section.

11.1 Enabling the Web server

You enable the Web server in STEP 7 from Device Configuration for the CPU to which you intend to connect.

To enable the Web server, follow these steps:

1. Select the CPU in the Device Configuration view.
2. In the inspector window, select "Web server" from the CPU properties.
3. Select the check box for "Enable Web server on this module".
4. If you require secure access to the standard web pages, select the "Permit access only with HTTPS" check box.

After you download the device configuration, you can use the standard Web pages to access the CPU. If you select "Enable" for "Automatic update", the pages refresh every ten seconds.

If you created user-defined Web pages, you can access them from the standard Web page menu.

Note

If a "Download in RUN" (Page 650) is in progress, standard and user-defined Web pages do not update data values or permit you to write data values until the download is complete. Any attempts to write data values while the download is in progress are discarded.

11.2 Standard web pages

11.2.1 Accessing the standard Web pages from the PC

To access the S7-1200 standard Web pages from a PC, follow these steps:

1. Ensure that the S7-1200 and the PC are on a common Ethernet network or are connected directly to each other with a standard Ethernet cable.
2. Open a Web browser and enter the URL "http://ww.xx.yy.zz", where "ww.xx.yy.zz" corresponds to the IP address of the S7-1200 CPU.

The Web browser opens the Introduction page.

Note

If your Internet access prevents direct connection to an IP address, see your IT administrator. Your Web environment or operating system might also impose other constraints (Page 488).

Alternatively, you can address your Web browser to a specific standard Web page. To do so, enter the URL in the form "http://ww.xx.yy.zz/<page>.html", where <page> corresponds to one of the standard Web pages:

- start (Page 479) - general information about the CPU
- identification (Page 480) - detailed information about the CPU including serial, order, and version numbers
- module (Page 481) - information about the modules in the local rack
- communication (Page 483) - information about the network addresses, physical properties of the communication interfaces, and communication statistics
- diagnostic (Page 480) - the diagnostic buffer
- variable (Page 484) - CPU variables and I/O, accessible by address or PLC tag name
- datalog (Page 486) - data log files stored internally in the CPU or on a memory card
- index (Page 478) - introduction page to enter the standard Web pages

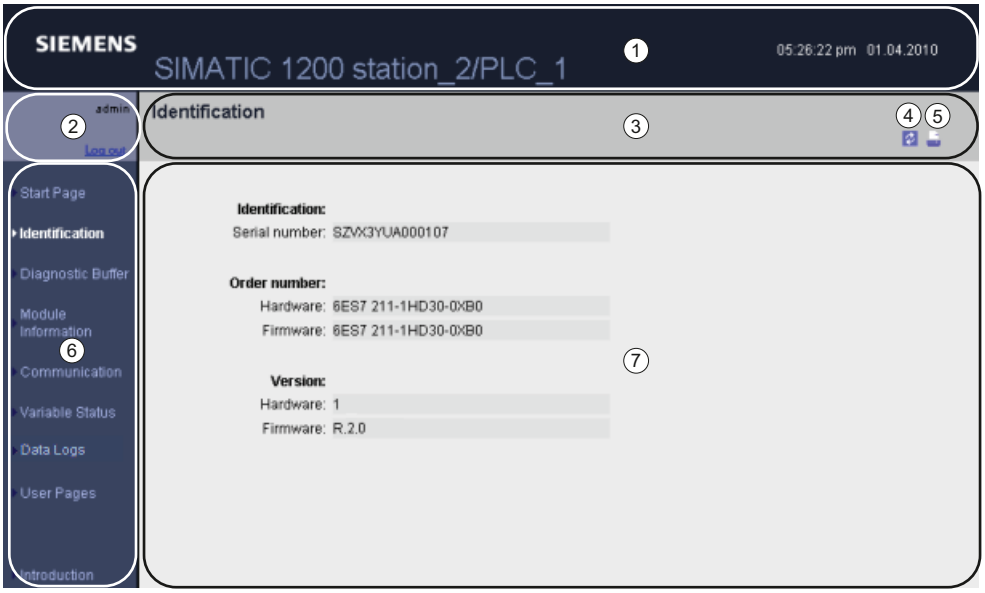
For example, if you enter "http://ww.xx.yy.zz/communication.html", the browser will display the communication page.

Secure access

You can use https:// instead of http:// for secure access to the standard Web pages. When you connect to the S7-1200 with https://, the website encrypts the session with a digital certificate. The data is securely transmitted and not accessible for anyone to view. You will typically get a security warning that you can confirm with "Yes" to proceed to the standard Web pages. To avoid the security warning with each secure access, you can import the Siemens software certificate to your Web browser (Page 490).

11.2.2 Layout of the standard Web pages

Each of the standard Web pages has a common layout with navigational links and page controls as shown below:



① Web server header

② Log in or log out

③ Standard Web page header with name of the page that you are viewing. This example is the CPU Identification page. Some of the standard Web pages, such as module information, also display a navigation path here if multiple screens of that type can be accessed.

④ Refresh icon: for pages with automatic refresh, enables or disables the automatic refresh function; for pages without automatic refresh, causes the page to update with current data

⑤ Print icon: prepares and displays a printable version of the information available from the displayed page

⑥ Navigation area to switch to another page

⑦ Content area for specific standard Web page that you are viewing. This example is the CPU Identification page.

Note

Printing standard Web pages

When printing standard Web page content, note that the printed contents can sometimes differ from the displayed page. For example, a print copy of the Diagnostic buffer page might contain new diagnostic entries that are not shown on the Diagnostic buffer page display. If automatic refresh is not enabled, the page display shows the diagnostic events at the time the page was initially displayed and the print copy contains the diagnostic events at the time the print function was executed.

Logging in

No log in is required to view the data in the standard Web pages. To perform certain actions such as changing the operating mode of the controller, or writing values to memory, you must log in as the "admin" user.

The log in frame is near the upper left corner on each page.

To log in as the "admin" user, follow these steps:

1. Enter "admin" for the Name field.
2. Enter the CPU password if one is configured in the Password field; otherwise, press the Enter key.

You are now logged in as the "admin" user.

If you encounter any errors logging in, return to the Introduction page (Page 478) and download the Siemens security certificate (Page 490). You can then log in with no errors.

Logging out

To log out the "admin" user, simply click the "Log out" link from any page.

You can continue to access and view standard Web pages when not logged in, but you cannot perform the actions that are restricted to the "admin" user. Each of the standard Web page descriptions defines the actions, if any, that require the "admin" log in.

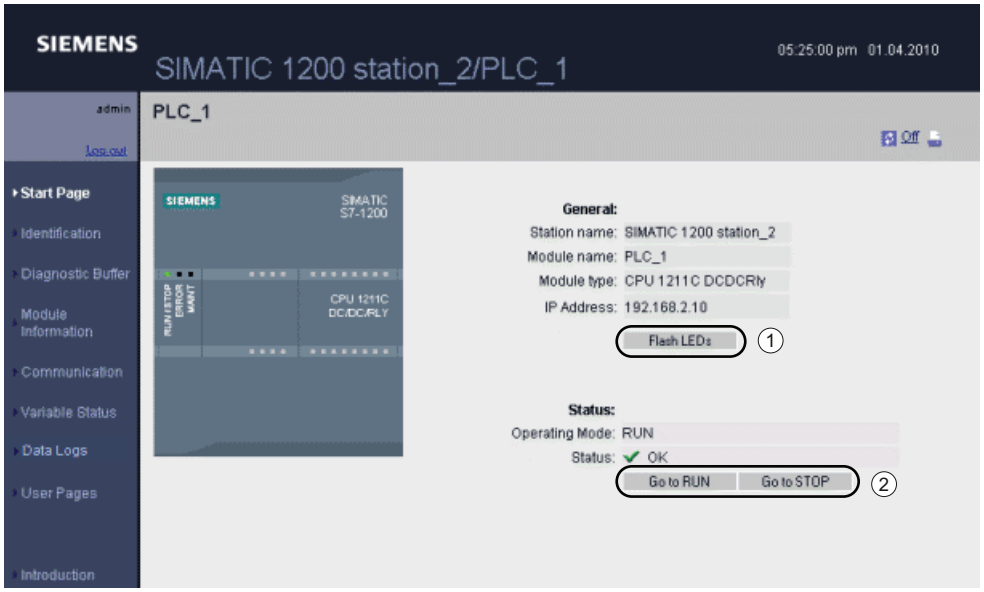
11.2.3 Introduction

The Introduction page is the welcome screen for entry into the S7-1200 standard Web pages.

From this page, you click "Enter" to access the S7-1200 standard Web pages. At the top of the screen are links to useful Siemens Web sites, as well as a link to download the Siemens security certificate (Page 490).

11.2.4 Start

The Start page displays a representation of the CPU to which you are connected and lists general information about the CPU. If you log in as the "admin" user, you can also change the operating mode of the CPU and flash the LEDs.



① and ② Buttons for flashing LEDs, and changing operating mode only appear on the Start page when you log in as the "admin" user.

11.2.5 Identification

The Identification page displays identifying characteristics of the CPU:

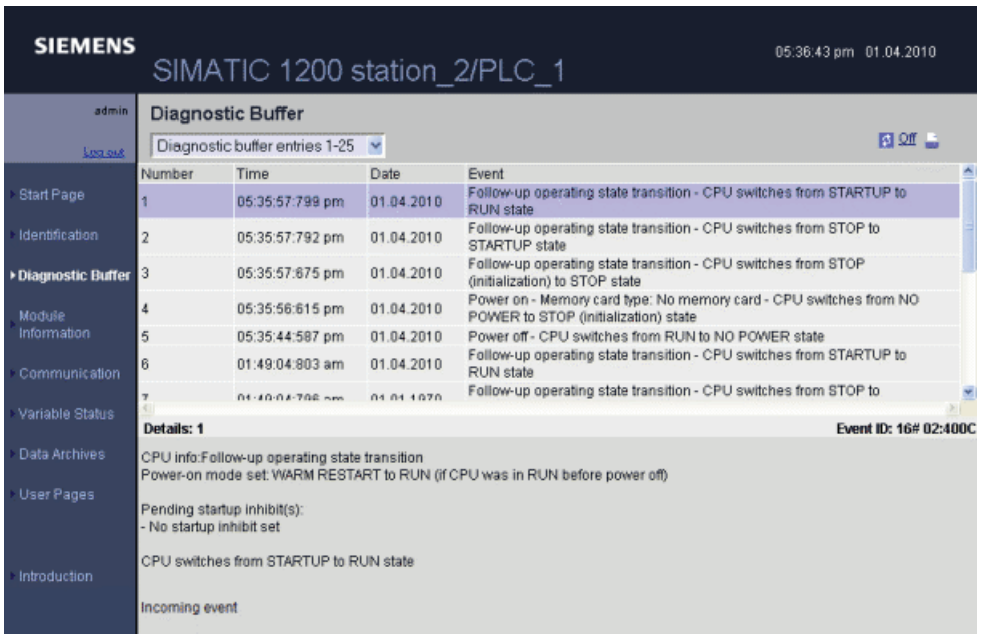
- Serial number
- Order numbers
- Version information

The Identification page does not vary with the "admin" login.

11.2.6 Diagnostic Buffer

The diagnostic buffer page displays diagnostic events. From the selector, you can choose what range of diagnostic buffer entries to display, either 1 to 25 or 26 to 50. The top part of the page displays those entries with the CPU time and date of when the event occurred. The times are CPU times, which correspond to the Time of day and Time zone setting in the device configuration for the CPU. The CPU time is not necessarily the same as the local time.

From the top part of the page, you can select any individual entry to show detailed information about that entry in the bottom part of the page.

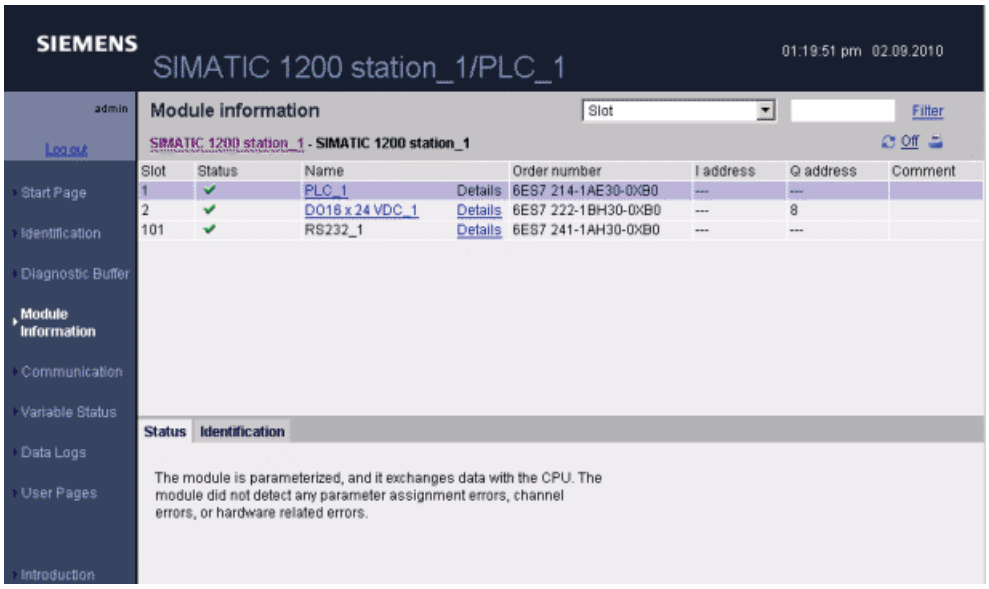


The Diagnostic buffer page does not vary with the "admin" login.

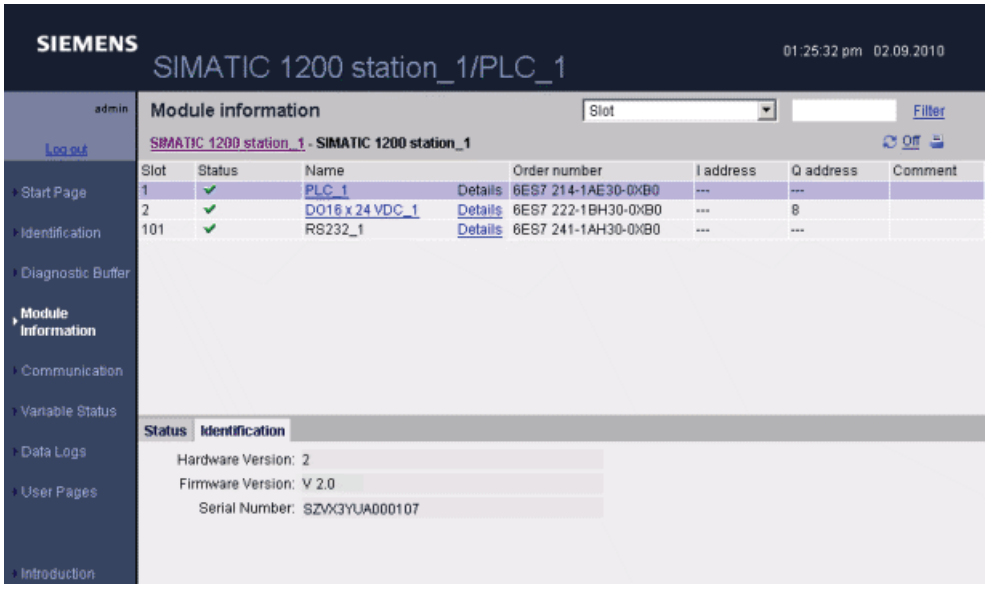
11.2.7 Module Information

The module information page provides information about all the modules in the local rack. The top section of the screen shows a summary of the modules, and the bottom section shows status and identification of the selected module.

Status display

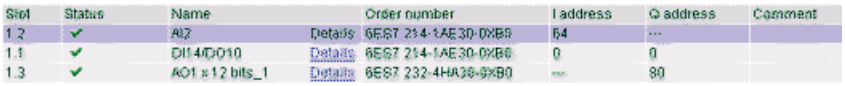


Identification display

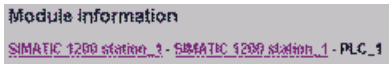


Drilling down

You can select a link in the top section to drill down to the module information for that particular module. Modules with submodules have links for each submodule. The type of information that is displayed varies with the module selected. For example, the module information dialog initially displays the name of the SIMATIC 1200 station, a status indicator, and a comment. If you drill down to the CPU, the module information displays the name of the digital and analog inputs and outputs that the CPU model provides (for example, "DI14/DO10", "AI2"), addressing information for the I/O, status indicators, slot numbers, and comments.

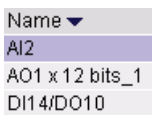


As you drill down, the module information page shows the path you have followed. You can click any link in this path to return to a higher level.



Sorting fields

When the list displays multiple modules, you can click the column header of a field to sort it either up or down by that field.



Filtering the module information

You can filter any field in the module information list. From the drop-down list, select the field name for which you want to filter the data. Enter text in the associated text box and click the Filter link. The list updates to show you modules that correspond to your filtering criteria.

Status information

The status tab in the bottom section of the module information page displays a description of the current status of the module that is selected in the top section.

Identification

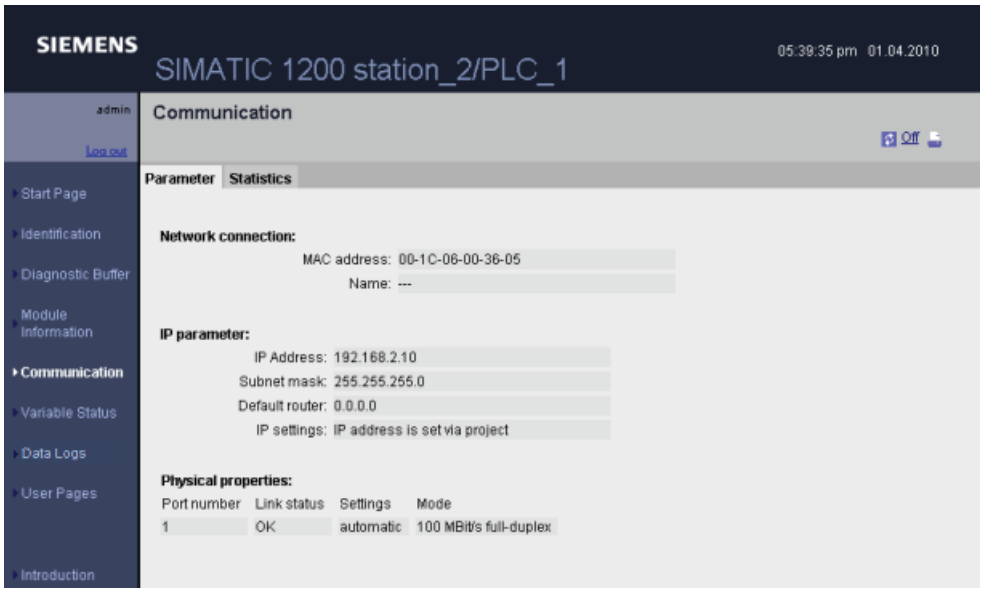
The identification tab displays the serial number and revision numbers of the selected module.

The module information page does not vary with the "admin" login.

11.2.8 Communication

The communication page displays the parameters of the connected CPU, and communications statistics. The Parameter tab shows the MAC address of the CPU, the IP address and IP settings of the CPU, and physical properties. The Statistics tab shows send and receive communication statistics.

Communication: Parameter display



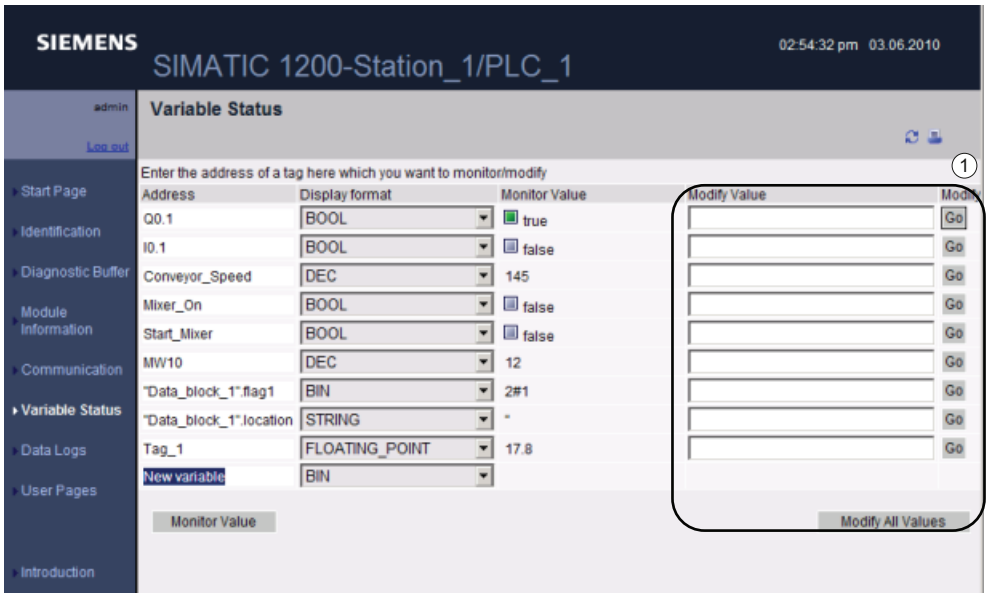
Communication: Statistics display

The communication page does not vary with the "admin" login.

11.2.9 Variable Status

The Variable Status page allows you to view any of the I/O or memory data in your CPU. You can enter a direct address (such as I0.0), a PLC tag name, or a tag from a specific data block. For data block tags, you enclose the data block name in double quotation marks. For each monitor value you can select a display format for the data. You can continue entering and specifying values until you have as many as you want within the limitations for the page. The monitor values show up automatically and refresh by default, unless you click the "Off" icon in the upper right area of the page. When refresh is disabled, you can click "On" to re-enable automatic refresh.

With the "admin" log in, you can also modify data values. Enter any values that you wish to set in the appropriate "Modify Value" field. Click the "Go" button beside a value to write that value to the CPU. To modify a variable of data type STRING, you must enclose the string in single quotation marks. You can also enter multiple values and click "Modify All Values" to write all of the values to the CPU.



① The "Modify Value" functionality is only visible and accessible when you are logged in as the "admin" user.

If you leave the Variable Status page and return, the Variable Status page does not retain your entries. You can bookmark the page and return to the bookmark to see the same entries. If you do not bookmark the page, you must re-enter the variables.

Note

Be aware of the following issues when using the standard Variable Status page:

- The Variable Status page does not allow you to modify a string longer than 198 characters.
- When using exponential notation to enter a value for a Real or LReal data type in the Variable Status page:
  - To enter a real-number value (Real or LReal) with a positive exponent (such as +3.402823e+25), do **not** enter the "+" for the exponent. Instead of entering "+3.402823e+25", you must enter the value in the following format:

    +3.402823e25
  - To enter real-number value (Real or LReal) with a negative exponent, enter the value as follows:

    +3.402823e-25
- The Variable Status page supports only 15 digits for an LReal value (regardless of the location of the decimal point). Entering more than 15 digits creates a rounding error.

Limitations on the Variable Status page:

- The maximum number of variable entries per page is 50.
- The maximum number of characters for the URL corresponding to the Variable Status page is 2083. You can see the URL that represents your current variable page in the address bar of your browser.
- For the character display format, the page displays hexadecimal values if the actual CPU values are not valid ASCII characters as interpreted by the browser.

Note

If a tag name displays special characters such that it is rejected as an entry on the Variable Status page, you can enclose the tag name in double quotation marks. In most cases, the Variable Status page will then recognize the tag name.

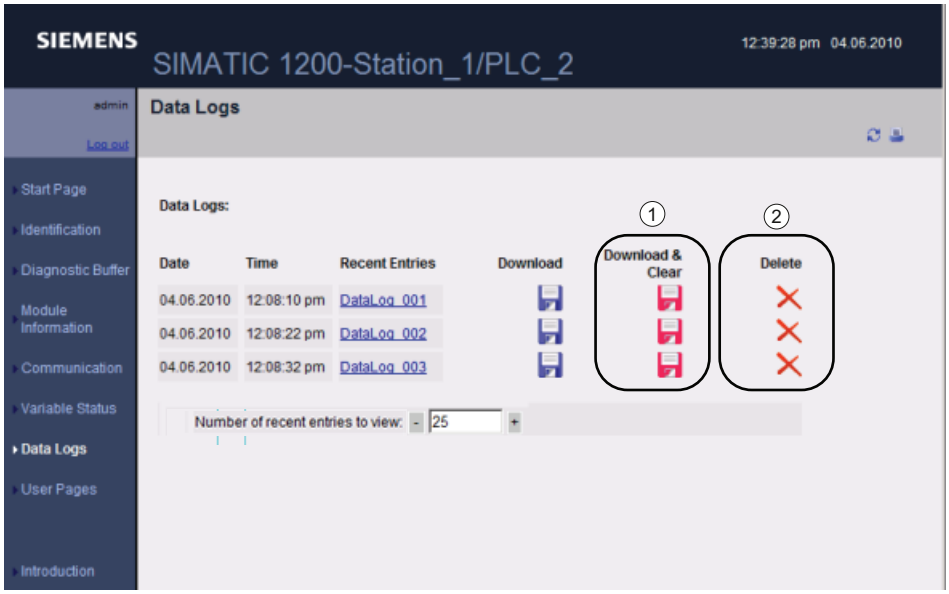
11.2.10 Data Logs

The Data Logs page allows you to view or download a specified number of data log entries. With the "admin" log in, you can also clear these entries after downloading them, or you can delete them. The Web server downloads data logs to your PC in comma-separated values (CSV) file format.

Note

Time stamp for data logs are shown in system time and not in local time

The CPU writes the time stamps for the data log records in system time (Coordinated Universal Time - UTC). The standard "Data Logs" page of the Web server displays the time stamps for the data logs in system time.



① The "Download & Clear" option is not available if you are not logged in as the "admin" user.

② The "Delete" option is not available if you are not logged in as the "admin" user.

Note

The data log file is in USA/UK comma-separated values format (CSV). To open it in Microsoft Excel on non-USA/UK systems, you must import it into Microsoft Excel with specific settings (Page 491).

Recent entries: Downloading a specified number of recent data records

Set the maximum number of recent records to download and then click the Data log name to initiate a download of the specified number of records. In the output .csv file, the data records are sorted in decreasing record number order. You will be prompted by Microsoft Windows to open or save the log file.

By default, the Data Logs page shows the most recent 25 entries of a log file, regardless of how many entries are actually in the log. You can change this value in the "Maximum most recent entries to read" field by entering a number or by using the + or - button to increment or decrement the value. The maximum number of entries per file that you can display or save is 50.

Downloading a log file that contains all data records

To download an entire log file, click the Download icon corresponding to a specific log file. You will be prompted by Microsoft Windows to open or save the log file.

In the output .csv file, all data records are included and sorted in increasing record number order unless the data log is full and older records (lower record number) are being overwritten by later records (higher record number).

Downloading and clearing a log file

To download a log file and then clear all the data records, you must be logged in as the "admin" user. Then click the "Download & Clear" icon corresponding to a specific log file. You will be prompted by Microsoft Windows to open or save the log file.

After the download is complete, a new "//END" line is inserted after the header record of the Data log file stored in the PLC. This effectively clears the Data log for future internal PLC processing, but subsequent downloads of this file will have new data records inserted above the first "//END" line.

Note

Data log .csv file "//END" marker

The "//END" .csv file end marker is only used for the first ((max records) -1) records to mark the logical end of the file. Behind the logical end, the file may contain data which may be interpreted by Excel as additional data records. You should search for the first "//END" then delete it and all records below it. If the logical end marker is not present, you can sort the data rows using the record number.

Deleting a log file

To delete a log file, you must be logged in as the "admin" user. Then click the Delete icon that corresponds to a specific log file. The Web server then deletes the selected log file.

Additional information

For information on programming with the Data log instructions, see the Data logging (Page 291) chapter.

11.2.11 Constraints

The following IT considerations can affect your use of the Web server:

- Typically, you must use the IP address of the CPU to access the standard Web pages or user-defined Web pages. If your Web browser does not allow connecting directly to an IP address, see your IT administrator. If your local policies support DNS, you can connect to the IP address through a DNS entry to that address.
- Firewalls, proxy settings, and other site-specific restrictions can also restrict access to the CPU. See your IT administrator to resolve these issues.
- The standard Web pages use JavaScripts and cookies. If JavaScripts or cookies are disabled in your Web browser, enable them. If you cannot enable them, some features will be restricted (Page 489). Use of JavaScripts and cookies in user-defined Web pages is optional. If used, they must be enabled in your browser.

- Secure Sockets Layer (SSL) is supported by the Web server. You can access the standard Web pages and user-defined Web pages with an URL of either http://ww.xx.yy.zz or https://ww.xx.yy.zz, where "ww.xx.yy.zz" represents the IP address of the CPU.
- Siemens provides a security certificate for secure access to the Web server. From the Introduction standard Web page (Page 478), you can download and import the certificate into the Internet options of your Web browser (Page 490). If you choose to not import the certificate, you will get a security verification prompt every time you access the Web server with https://.

11.2.11.1 Features restricted when JavaScript is disabled

The standard Web pages are implemented using HTML, JavaScripts, and cookies. If your site restricts the use of JavaScripts and cookies, then enable them for the pages to function properly. If you cannot enable JavaScripts for your Web browser, the features controlled by JavaScripts cannot run.

General

The pages do not update dynamically. You must manually refresh the page with the Refresh icon (Page 476) to view fresh data.

Diagnostic Buffer page

- Displaying the event details: With JavaScript, you select a row in the diagnostic buffer to see the details in the bottom section. Without JavaScript, you must click the event field hyperlink of a diagnostic buffer entry to see the event data in the bottom section.
- Changing the range of diagnostic buffer entries to view: With JavaScript, you use the drop-down list at the top to select the range of diagnostic buffer entries to view, and the page automatically updates. Without JavaScript, you use the drop-down list at the top to select the range of diagnostic buffer entries to view, but you must then click the "Go" link to update the diagnostic buffer page with the range you selected from the drop-down list.

Note that the "Go" and the event field hyperlinks are only visible when JavaScript is not enabled. They are not necessary and therefore are not present when JavaScript is enabled.

Note

The Opera V11.0 browser does not support the "Go" button or hyperlinked diagnostic entries. With Opera V11.0, you cannot access event details or change the range if you have disabled JavaScript.

Module Information page

- You cannot filter the data.
- You cannot sort fields.

Variable Status page

- After you enter each variable, you must manually set the focus to the "New variable" row to enter a new variable.
- Selecting a display format does not automatically change the data value display to the selected format. You must click the "Monitor value" button to refresh the display with the new format.

Data Logs page

- You cannot click a file name under "Recent entries" to open or save a log file. You can, however, use the Download icon for the same functionality.
- The Data Logs page does not refresh.
- The "+" and "-" buttons to increment and decrement the number of recent entries have no effect.
- Entering a value directly in the number of recent entries does not set the number of entries. If you attempt to enter a value in this field from Mozilla Firefox, the display goes blank. You must reselect "Data Logs" from the navigation pane to restore the Data Logs display. The number of recent entries field remains unchanged.

Note that you can leave the Data Logs page and re-enter to get the most recent 25 entries.

11.2.11.2 Features restricted when cookies are not allowed

If your Web browser does not allow cookies you cannot log in with the "admin" user name.

11.2.11.3 Importing the Siemens security certificate

You can import the Siemens security certificate into your Internet options so that you won't be prompted for security verification when you enter https://ww.xx.yy.zz in your Web browser, where "ww.xx.yy.zz" is the IP address of the CPU. If you use an http:// URL and not an https:// URL, then you do not need to download and install the certificate.

Downloading the certificate

You use the "download certificate" link from the Introduction page (Page 478) to download the Siemens security certificate to your PC. The procedure varies according to which Web browser you use:

Importing the certificate to Internet Explorer

1. Click the "download certificate" link from the Introduction page. A "File Download - Security Warning" dialog pops up.
2. From the "File Download - Security Warning" dialog, click "Open" to open the file. A "Certificate" dialog appears.
3. From the "Certificate" dialog, click the "Install Certificate" button to launch the Certificate Import Wizard.
4. Follow the dialogs of the "Certificate Import Wizard" to import the certificate, letting the operating system automatically select the certificate store.

Importing the certificate to Mozilla Firefox

1. Click the "download certificate" link from the Intro page. An "Opening MiniWebCA_Cer.crt" dialog pops up.
2. Click "Save file" from the "Opening MiniWebCA_Cer.crt" dialog. A "Downloads" dialog appears.
3. From the "Downloads" dialog, double-click "MiniWebCA_Cer.crt". If you have attempted the download more than once, multiple copies show up. Just double-click any one of the "MiniWebCA_Cer.crt". entries.
4. Click "OK" if prompted to open an executable file.
5. Click "Open" on the "Open File - Security Warning" dialog if it appears. A "Certificate" dialog appears.
6. On the "Certificate" dialog, click the "Install Certificate" button.
7. Follow the dialogs of the "Certificate Import Wizard" to import the certificate, letting the operating system automatically select the certificate store.
8. If the "Security Warning" dialog appears, click "Yes" to confirm installation of the certificate.

Other browsers

Follow the conventions of your Web browser to import and install the Siemens certificate.

After you have installed the Siemens security certificate "SIMATIC CONTROLLER" in the Internet options for your Web browser content, you will not be required to verify a security prompt when you access the Web server with https:// ww.xx.yy.zz.

11.2.11.4 Importing CSV format data logs to non-USA/UK versions of Microsoft Excel

Data log files are in the comma-separated values (CSV) file format. You can open these files directly in Microsoft Excel from the Data Logs page when your system is running the USA or UK version of Microsoft Excel. In other countries, however, this format is not widely used because commas occur frequently in numerical notation.

To open a data log file that you have saved, follow these steps for non USA/UK versions of Microsoft Excel:

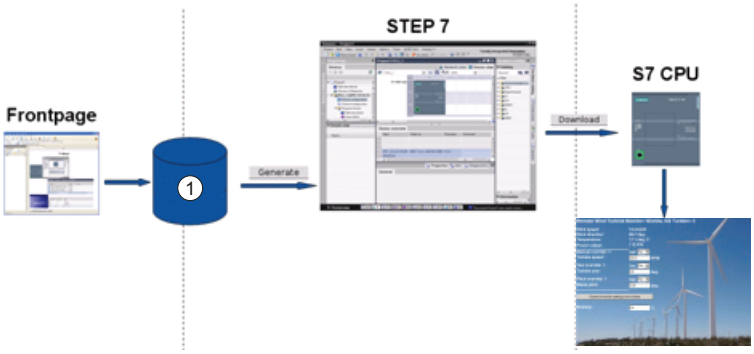
1. Open Excel and create an empty workbook.
2. From the "Data > Import External Data" menu, select the "Import Data" command.
3. Navigate to and select the data log file you want to open. The Text Import Wizard starts.
4. From the Text Import Wizard, change the default option for "Original data type" from "Fixed width" to "Delimited".
5. Click the Next button.
6. From the Step 2 dialog, select the "Comma" check box to change the delimiter type from "Tab" to "Comma".
7. Click the Next button.
8. From the Step 3 dialog, you can optionally change the Date format from MDY (month/day/year) to another format.
9. Complete the remaining steps of the Text Import Wizard to import the file.

11.3 User-defined web pages

The S7-1200 Web server also provides the means for you to create your own application-specific HTML pages that incorporate data from the PLC. You create these pages using the HTML editor of your choice and download them to the CPU where they are accessible from the standard Web page menu. This process involves several tasks:

- Creating HTML pages with an HTML editor, such as Microsoft Frontpage (Page 493)
- Including AWP commands in HTML comments in the HTML code (Page 494):The AWP commands are a fixed set of commands that Siemens provides for accessing CPU information.
- Configuring STEP 7 to read and process the HTML pages (Page 507)
- Generating blocks from the HTML pages (Page 507)
- Programming STEP 7 to control the use of the HTML pages (Page 508)
- Compiling and downloading the blocks to the CPU (Page 509)
- Accessing the user-defined Web pages from your PC (Page 510)

This process is illustrated below:



① HTML files with embedded AWP commands

11.3.1 Creating HTML pages

You can use the software package of your choice to create your own HTML pages for use with the Web server. Be sure that your HTML code is compliant to the HTML standards of the W3C (World Wide Web Consortium). STEP 7 does not perform any verification of your HTML syntax.

You can use a software package that lets you design in WYSIWYG or design layout mode, but you need to be able to edit your HTML code in pure HTML form. Most Web authoring tools provide this type of editing; otherwise, you can always use a simple text editor to edit the HTML code. Include the following line in your HTML page to set the charset for the page to UTF-8:

```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```

Also be sure to also save the file from the editor in UTF-8 character encoding:

You use STEP 7 to compile everything in your HTML pages into STEP 7 data blocks. These data blocks consist of one control data block that directs the display of the Web pages and one or more fragment data blocks that contain the compiled Web pages. Be aware that extensive sets of HTML pages, particularly those with lots of images, require a significant amount of load memory space (Page 510) for the fragment DBs. If the internal load memory of your CPU is not sufficient for your user-defined Web pages, use a memory card (Page 99) to provide external load memory.

To program your HTML code to use data from the S7-1200, you include AWP commands (Page 494) as HTML comments. When finished, save your HTML pages to your PC and note the folder path where you save them.

Refreshing user-defined Web pages

User-defined Web pages do not automatically refresh. It is your choice whether to program the HTML to refresh the page or not. For pages that display PLC data, refreshing periodically keeps the data current. For HTML pages that serve as forms for data entry, refreshing can interfere with the user entering data. If you want your entire page to automatically refresh, you can add this line to your HTML header, where "10" is the number of seconds between refreshes:

```
<meta http-equiv="Refresh" content="10">
```

You can also use JavaScripts or other HTML techniques to control page or data refreshing. For this, refer to documentation on HTML and JavaScript.

11.3.2 AWP commands supported by the S7-1200 Web server

The S7-1200 Web server provides AWP commands that you embed in your user-defined Web pages as HTML comments for the following purposes:

- Reading variables (Page 495)
- Writing variables (Page 496)
- Reading special variables (Page 497)
- Writing special variables (Page 499)
- Defining enum types (Page 501)
- Assigning variables to enum types (Page 501)
- Creating fragment data blocks (Page 503)

General syntax

Except for the command to read a variable, the AWP commands are of the following syntax:

```
<!-- AWP_ <command name and parameters> -->
```

You use the AWP commands in conjunction with typical HTML form commands to write to variables in the CPU.

The descriptions of the AWP commands in the following pages use the following conventions:

- Items enclosed in brackets [ ] are optional.
- Items enclosed in angle brackets < > are parameter values to be specified.
- Quotation marks are a literal part of the command. They must be present as indicated.
- Special characters in tag or data block names, depending on usage, must be escaped or enclosed in quotation marks (Page 505).

Use a text editor or HTML editing mode to insert AWP commands into your pages.

AWP command summary

The details for using each AWP command are in the topics to follow, but here is a brief summary of the commands:

Reading variables
```
:=<Varname>:
```

Writing variables
```
<!-- AWP_In_Variable Name='<Varname1>' [Use='<Varname2>'] ... -->
```

This AWP command merely declares the variable in the Name clause to be writable. Your HTML code performs writes to the variable by name from <input>, <select>, or other HTML statements within an HTML form.

Reading special variables
```
<!-- AWP_Out_Variable Name='<Type>:<Name>' [Use='<Varname>'] -->
```

Writing special variables
```
<!-- AWP_In_Variable Name='<Type>:<Name>' [Use='<Varname>']-->
```

Defining enum types
```
<!--
 AWP_Enum_Def Name='<Enum type name>' Values='<Value>, <Value>,... '
 -->
```

Referencing enum types
```
<!-- AWP_Enum_Ref Name='<VarName>' Enum='<EnumType>' -->
```

Creating fragments
```
<!-- AWP_Start_Fragment Name='<Name>' [Type=<Type>][ID=<id>] -->
```

Importing fragments
```
<!-- AWP_Import_Fragment Name='<Name>' -->
```

11.3.2.1 Reading variables

User-defined Web pages can read variables (PLC tags) from the CPU.

Syntax

```
:=<Varname>:
```

Parameters

<Varname>	The variable to be read, which can be a PLC tag name from your STEP 7 program, a data block tag, I/O, or addressable memory. For memory or I/O addresses or alias names (Page 505), do not use quotation marks around the tag name. For PLC tags, use double quotation marks around the tag name. For data block tags, enclose the block name only in double quotation marks. The tag name is outside of the quotation marks. Note that you use the data block name and not a data block number.

Examples

```
:="Conveyor_speed"::="My_Data_Block".flag1:
:=I0.0:
:=MW100:
```

Example reading an aliased variable

```
<!--AWP_Out_Variable Name='flag1' Use='"My_Data_Block".flag1' -->
:=flag1:
```

Note

Defining alias names for PLC tags and data block tags is described in the topic Using an alias for a variable reference (Page 500).

If a tag name or data block name includes special characters, you must use additional quotation marks or escape characters as described in the topic Handling tag names that contain special characters (Page 505).

11.3.2.2 Writing variables

User-defined pages can write data to the CPU. This is accomplished by using an AWP command to identify a variable in the CPU to be writable from the HTML page. The variable must be specified by PLC tag name or data block tag name. You can declare multiple variable names in one statement. To write the data to the CPU, you use standard HTTP POST commands.

A typical usage is to design a form in your HTML page with text input fields or select list choices that correspond to writable CPU variables. As with all user-defiined pages, you then generate the blocks from STEP 7 such that they are included in your STEP 7 program. When an admin user subsequently accesses this page and types data into the input fields or selects a choice from a select list, the Web server converts the input to the appropriate data type for the variable, and writes the value to the variable in the CPU. Note that the name clause for HTML input fields and HTML select lists uses syntax typical for the name clause of the AWP_In_Variable command. Typically enclose the name in single quotation marks and if you reference a data block, enclose the data block name in double quotation marks.

For form management details, refer to documentation for HTML.

Syntax

```
<!-- AWP_In_Variable Name='<Varname1>' [Use='<Varname2>'] ... -->
```

Parameters

<Varname1>	If no Use clause is provided, Varname1 is the variable to be written. It can be a PLC tag name from your STEP 7 program or a tag from a specific data block. If a Use clause is provided, Varname1 is an alternate name for the variable referenced in <Varname2> (Page 500). It is a local name within the HTML page.
<Varname2>	If a Use clause is provided, Varname2 is the variable to be written. It can be a PLC tag name from your STEP 7 program or a tag from a specific data block.

For both Name clauses and Use clauses, the complete name must be enclosed in single quotation marks. Within the single quotes, use double quotation marks around a PLC tag and double quotation marks around a data block name. The data block name is within the double quotes but not the data block tag name. Note that for data block tags, you use the name of the block and not a data block number.

Examples using HTML input field

```
<!-- AWP_In_Variable Name='"Target_Level"' -->
<form method="post">
<p>Input Target Level: <input name='"Target_Level"' type="text" />
</p>
</form>

<!-- AWP_In_Variable Name='"Data_block_1".Braking' -->
<form method="post">
<p>Braking: <input name='"Data_block_1".Braking' type="text" />
%</p>
</form>

<!-- AWP_In_Variable Name='"Braking"' Use='"Data_block_1".Braking' -
->
<form method="post">
<p>Braking: <input name='"Braking"' type="text" /> %</p>
</form>
```

Example using HTML select list

```
<!-- AWP_In_Variable Name='"Data_block_1".ManualOverrideEnable'-->
<form method="post">
<select name='"Data_block_1".ManualOverrideEnable'>
<option value=:"Data_block_1".ManualOverrideEnable:> </option>
<option value=1>Yes</option>
<option value=0>No</option>
</select><input type="submit" value="Submit setting" /></form>
```

Note

Only an admin user can write data to the CPU. The commands are ignored if the user has not logged in as the admin user.

If a tag name or data block name includes special characters, you must use additional quotation marks or escape characters as described in the topic "Handling tag names that contain special characters (Page 505)".

11.3.2.3 Reading special variables

The Web server provides the ability to read values from the PLC to store in special variables in the HTTP response header. You might, for example, want to read a pathname from a PLC tag to redirect the URL to another location using the HEADER:Location special variable.

Syntax

```
<!-- AWP_Out_Variable Name='<Type>:<Name>' [Use='Varname>'] -->
```

Parameters

<Type>	The type of special variable and is one of the following: HEADER COOKIE_VALUE COOKIE_EXPIRES
<Name>	Refer to HTTP documentation for a list of all the names of HEADER variables. A few examples are listed below: Status: response code Location: path for redirection Retry-After: how long service is expected to be unavailable to the requesting client For types COOKIE_VALUE and COOKIE_EXPIRES, <Name> is the name of a specific cookie. COOKIE_VALUE:name: value of the named cookie COOKIE_EXPIRES:name: expiration time in seconds of named cookie The Name clause must be enclosed in single or double quotation marks. If no Use clause is specified, the special variable name corresponds to a PLC tag name. Enclose the complete Name clause within single quotation marks and the PLC tag in double quotation marks. The special variable name and PLC tag name must match exactly.
<Varname>	Name of the PLC tag or data block tag for the variable to be read into The Varname must be enclosed in single quotation marks. Within the single quotes, use double quotation marks around a PLC tag or data block name. The data block name is within the double quotes but not the data block tag name. Note that for data block tags, you use the name of the block and not a data block number.

Examples

```
<!-- AWP_Out_Variable Name='"HEADER:Status"' -->
```

In this example, the HTTP special variable "HEADER:Status" receives the value of the PLC tag "HEADER:Status". The name in the PLC tag table must match the name of the special variable exactly if no Use clause is specified.

```
<!-- AWP_Out_Variable Name='HEADER:Status' Use='"Status"' -->
```

In this example, the HTTP special variable "HEADER:Status" receives the value of the PLC tag "Status".

If a tag name or data block name includes special characters, you must use additional quotation marks or escape characters as described in the topic Handling tag names that contain special characters (Page 505).

11.3.2.4 Writing special variables

The Web server provides the ability to write values to the CPU from special variables in the HTTP request header. For example, you can store information in STEP 7 about the cookie associated with a user-defined Web page, the user that is accessing a page, or header information. The Web server provides access to specific special variables that you can write to the CPU when logged in as the admin user.

Syntax

```
<!-- AWP_In_Variable Name='<Type>:<Name>' [Use='<Varname>']-->
```

Parameters

<Type>	The type of special variable and is one of the following: HEADER SERVER COOKIE_VALUE
<Name>	Specific variable within the types defined above, as shown in these examples: HEADER:Accept: content types that are acceptable HEADER:User-Agent: information about the user agent originating the request. SERVER:current_user_id: id of the current user; 0 if no user logged in SERVER:current_user_name: name of the current user COOKIE_VALUE:<name>: value of the named cookie Enclose the Name clause in single quotation marks. If no Use clause is specified, the special variable name corresponds to a PLC variable name. Enclose the complete Name clause within single quotation marks and the PLC tag in double quotation marks. The special variable name must match the PLC tag name exactly. Refer to HTTP documentation for a list of all the names of HEADER variables.
<Varname>	The variable name in your STEP 7 program into which you want to write the special variable, which can be a PLC tag name, or a data block tag. The Varname must be enclosed in single quotation marks. Within the single quotes, use double quotation marks around a PLC tag or data block name. The data block name is within the double quotes but not the data block tag name. Note that for data block tags, you use the name of the block and not a data block number.

Examples

```
<!-- AWP_In_Variable Name='"SERVER:current_user_id"' -->
```

In this example, the Web page writes the value of the HTTP special variable "SERVER:current_user_id" to the PLC tag named "SERVER:current_user_id ".

```
<!-- AWP_In_Variable Name=SERVER:current_user_id' Use='"my_userid"'
-->
```

In this example, the Web page writes the value of the HTTP special variable "SERVER:current_user_id" to the PLC tag named "my_userid".

Note

Only an admin user can write data to the CPU. The commands are ignored if the user has not logged in as the admin user.

If a tag name or data block name includes special characters, you must use additional quotation marks or escape characters as described in the topic "Handling tag names that contain special characters (Page 505)".

11.3.2.5 Using an alias for a variable reference

You can use an alias in your user-defined Web page for an In_Variable or an Out_Variable. For example, you can use a different symbolic name in your HTML page than the one used in the CPU, or you can equate a variable in the CPU with a special variable. The AWP Use clause provides this capability.

Syntax

```
<-- AWP_In_Variable Name='<Varname1>' Use='<Varname2>' -->
<-- AWP_Out_Variable Name='<Varname1>' Use='<Varname2>' -->
```

Parameters

<Varname1>	The alias name or special variable name Varname1 must be enclosed in single or double quotation marks.
<Varname2>	Name of the PLC variable for which you want to assign an alias name. The variable can be a PLC tag, a data block tag, or a special variable. Varname2 must be enclosed in single quotation marks. Within the single quotes, use double quotation marks around a PLC tag, special variable, or data block name. The data block name is within the double quotes but not the data block tag name. Note that for data block tags, you use the name of the block and not a data block number.

Examples

```
<-- AWP_In_Variable Name='SERVER:current_user_id'
Use='"Data_Block_10".server_user' -->
```

In this example, the special variable SERVER:current_user_id is written to the tag "server_user" in data block "Data_Block_10".

```
<-- AWP_Out_Variable Name='Weight'
Use='"Data_Block_10".Tank_data.Weight' -->
```

In this example, the value in data block structure member Data_Block_10.Tank_data.Weight can be referenced simply by "Weight" throughout the rest of the user-defined Web page.

```
<-- AWP_Out_Variable Name='Weight' Use='"Raw_Milk_Tank_Weight"' -->
```

In this example, the value in the PLC tag "Raw_Milk_Tank_Weight" can be referenced simply by "Weight" throughout the rest of the user-defined Web page.

If a tag name or data block name includes special characters, you must use additional quotation marks or escape characters as described in the topic Handling tag names that contain special characters (Page 505).

11.3.2.6 Defining enum types

You can define enum types in your user-defined pages and assign the elements in an AWP command.

Syntax

```
<!-- AWP_Enum_Def Name='<Enum type name>' Values='<Value>,
<Value>,... ' -->
```

Parameters

<Enum type name>	Name of the enumerated type, enclosed in single or double quotation marks.
<Value>	<constant>:<name> The constant indicates the numerical value for the enum type assignment. The total number is unbounded. The name is the value assigned to the enum element.

Note that the entire string of enum value assignments is enclosed in single quotation marks, and each individual enum type element assignment is enclosed in double quotation marks. The scope of an enum type definition is global for the user-defined Web pages. If you have set up your user-defined Web pages in language folders (Page 522), the enum type definition is global for all pages in the language folder.

Example

```
<!-- AWP_Enum_Def Name='AlarmEnum' Values='0:"No alarms", 1:"Tank is
full", 2:"Tank is empty"' -->
```

11.3.2.7 Referencing CPU variables with an enum type

You can assign a variable in the CPU to an enum type. This variable can be used elsewhere in your user-defined Web page in a read operation (Page 495) or a write operation (Page 496). On a read operation, the Web server will replace the numerical value that is read from the CPU with the corresponding enum text value. On a write operation, the Web server will replace the text value with the integer value of the enumeration that corresponds to the text before writing the value to the CPU.

Syntax

```
<!-- AWP_Enum_Ref Name='<Varname>' Enum="<EnumType>" -->
```

Parameters

<Varname>	Name of PLC tag or data block tag to associate with the enum type, or the name of the alias name for a PLC tag (Page 500) if declared. Varname must be enclosed in single quotation marks. Within the single quotes, use double quotation marks around a PLC tag or data block name. Note that for data block tags, you use the name of the block and not a data block number. The data block name is within the double quotes but not the data block tag name.
<EnumType>	Name of the enumerated type, which must be enclosed in single or double quotation marks

The scope of an enum type reference is the current fragment.

Example declaration

```
<!-- AWP_Enum_Ref Name='"Alarm"' Enum="AlarmEnum" -->
```

Example usage in a variable read

```
<!-- AWP_Enum_Def Name='AlarmEnum' Values='0:"No alarms", 1:"Tank is
full", 2:"Tank is empty"' -->
<!-- AWP_Enum_Ref Name='"Alarm"' Enum="AlarmEnum" -->
...
<p>The current value of "Alarm" is :="Alarm":</p>
```

If the value of "Alarm" in the CPU is 2, the HTML page displays 'The current value of "Alarm" is Tank is empty' because the enum type definion (Page 501) assigns the text string "Tank is empty" to the numerical value 2.

Example usage in a variable write

```
<!-- AWP_In_Variable Name='"Alarm"' -->
<!-- AWP_Enum_Def Name='AlarmEnum' Values='0:"No alarms", 1:"Tank is
full", 2:"Tank is empty"' -->
<!-- AWP_Enum_Ref Name='"Alarm"' Enum='AlarmEnum' -->
...
<form method="POST">
<p><input type="hidden" name='"Alarm"' value="Tank is full" /></p>
<p><input type="submit" value='Set Tank is full' /><p>
</form>
```

Because the enum type defintion (Page 501) assigns "Tank is full" to the numerical value 1, the value 1 is written to the PLC tag named "Alarm" in the CPU.

Note that the Name clause in the AWP_In_Variable declaration must correspond exactly to the Name clause in the AWP_Enum_Ref declaration.

Example usage in a variable write with use of an alias

```
<!-- AWP_In_Variable Name='"Alarm"'
Use='"Data_block_4".Motor1.Alarm'-->
<!-- AWP_Enum_Def Name='AlarmEnum' Values='0:"No alarms", 1:"Tank is
full", 2:"Tank is empty"' -->
<!-- AWP_Enum_Ref Name='"Alarm"' Enum='AlarmEnum' -->
...
```

```
<form method="POST">
<p><input type="hidden" name='"Alarm"' value="Tank is full" /></p>
<p><input type="submit" value='Set Tank is full' /><p>
</form>
```

Because the enum type defintion (Page 501) assigns "Tank is full" to the numerical value 1, the value 1 is written to the alias "Alarm" which corresponds to the PLC tag named "Motor1.Alarm" in data block "Data_Block_4" in the CPU.

If a tag name or data block name includes special characters, you must use additional quotation marks or escape characters as described in the topic Handling tag names that contain special characters (Page 505).

11.3.2.8 Creating fragments

STEP 7 converts and stores user-defined Web pages as a control DB and fragment DBs when you click "Generate blocks" in the CPU Properties for the Web server. You can set up specific fragments for specific pages or for sections of specific pages. You can identify these fragments by a name and number with the "Start_Fragment" AWP command. Everything in the page following the AWP_Start_Fragment command belongs to that fragment until another AWP_Start_Command is issued or until end of file is reached.

Syntax

```
<!-- AWP_Start_Fragment Name='<Name>' [Type=<Type>][ID=<id>] -->
```

Parameters

<Name>	Text string: name of fragment DB Fragment names must begin with a letter or underscore and be comprised of letters, numeric digits, and underscores. The fragment name is a regular expression of the form: [a-zA-Z_][a-zA-Z_0-9]*
<Type>	"manual" or "automatic" manual: The STEP 7 program must request this fragment and can respond accordingly. Operation of the fragment must be controlled with STEP 7 and the control DB variables. automatic: The Web server processes the fragment automatically. If you do not specify the type parameter, the default is "automatic".
<id>	Integer identification number. If you do not specify the ID parameter, the Web server assigns a number by default. For manual fragments, set the ID to a low number. The ID is the means by which the STEP 7 program controls a manual fragment.

Manual fragments

If you create a manual fragment for a user-defined Web page or portion of a page, then your STEP 7 program must control when the fragment is sent. The STEP 7 program must set appropriate parameters in the control DB for a user-defined page under manual control and then call the WWW instruction with the control DB as modified. For understanding the structure of the control DB and how to manipulate individual pages and fragments, see the topic Advanced user-defined Web page control (Page 526).

11.3.2.9 Importing fragments

You can create a named fragment from a portion of your HTML code and then import that fragment elsewhere in your set of user-defined Web pages. For example, consider a set of user-defined Web pages that has a start page and then several other HTML pages accessible from links on the start page. Suppose each of the separate pages is to display the company logo on the page. You could implement this by creating a fragment (Page 503) that loads the image of the company logo. Each individual HTML page could then import this fragment to display the company logo. You use the AWP Import_Fragment command for this purpose. The HTML code for the fragment only exists in one fragment, but you can import this fragment DB as many times as necessary in as many Web pages as you choose.

Syntax

```
<!-- AWP_Import_Fragment Name='<Name>' -->
```

Parameters

<Name>	Text string: name of the fragment DB to be imported

Example

Excerpt from HTML code that creates a fragment to display an image:

```
<!-- AWP_Start_Fragment Name='My_company_logo' --><p><img
src="company_logo.jpg"></p>
```

Excerpt from HTML code in another .html file that imports the fragment that displays the logo image:

```
<!-- AWP_Import_Fragment Name='My_company_logo' -->
```

Both .html files (the one that creates the fragment and the one that imports it) are in the folder structure that you define when you configure the user-defined pages in STEP 7 (Page 507).

11.3.2.10 Combining definitions

When declaring variables for use in your user-defined Web pages, you can combine a variable declaration and an alias for the variable (Page 500). You can also declare multiple In_Variables in one statement and multiple Out_Variables in one statement.

Examples

```
<!-- AWP_In_Variable Name='"Level"', Name='"Weight"', Name='"Temp"'
-->
<--! AWP_Out_Variable Name='HEADER:Status', Use='"Status"',
         Name='HEADER:Location', Use="Location",
         Name='COOKIE_VALUE:name', Use="my_cookie" -->
<!-- AWP_In_Variable Name='Alarm' Use='"Data_block_10".Alarm' -->
```

11.3.2.11 Handling tag names that contain special characters

When specifying variable names in user-defined Web pages, you must take special care if tag names contain characters that have special meanings.

Reading variables

You use the following syntax to read a variable (Page 495):
:=<Varname>:

The following rules apply to reading variables:

- For variable names from the PLC tag table, enclose the tag name in double quotation marks.
- For variable names that are data block tags, enclose the data block name in double quotation marks. The tag is outside of the quotation marks.
- For variable names that are direct I/O addresses, memory addresses, or alias names, do not use quotation marks around the read variable.
- For tag names or data block tag names that contain a backslash, precede the backslash with another backslash.
- If a tag name or data block tag name contains a colon, less than sign, greater than sign, or ampersand define an alias that has no special characters for the read variable, and read the variable using the alias. Precede colons in tag names in a Use clause with a backslash.

Table 11- 1 Examples of Read variables

Data block name	Tag name	Read command
n/a	ABC:DEF	`<!--AWP_Out_Variable Name='special_tag' Use ='"ABC:DEF"' --> :=special_tag:`
n/a	T\	`:="T\\":`
n/a	A \B 'C :D	`<!--AWP_Out_Variable Name='another_special_tag' Use='"A \\B \'C :D"' --> :=another_special_tag:`
n/a	a<b	`<!--AWP_Out_Variable Name='a_less_than_b' Use='"a<b"' --> :=a_less_than_b:`
Data_block_1	Tag_1	`:="Data_block_1".Tag_1:`

Data block name	Tag name	Read command
Data_block_1	ABC:DEF	`<!-- AWP_Out_Variable Name='special_tag' Use='"Data_block_1".ABC\:DEF'--> :=special_tag:`
DB A' B C D$ E	Tag	`:="DB A' B C D$ E".Tag:`
DB:DB	Tag:Tag	`<!--AWP_Out_Variable Name='my_tag' Use ='"DB:DB".Tag\:Tag' --> :=my_tag:`

Name and Use clauses

The AWP commands AWP_In_Variable, AWP_Out_Variable, AWP_Enum_Def, AWP_Enum_Ref, AWP_Start_Fragment and AWP_Import_Fragment have Name clauses. HTML form commands such as <input> and <select> also have name clauses. AWP_In_Variable and AWP_Out_Variable can additionally have Use clauses. Regardless of the command, the syntax for Name and Use clauses regarding the handling of special characters is the same:

- The text you provide for a Name or Use clause must be enclosed within single quotation marks. If the enclosed name is a PLC tag or Data block name, use single quotation marks for the full clause.
- Within a Name or Use clause, data block names and PLC tag names must be enclosed within double quotation marks.
- If a tag name or Data block name includes a single quote character or backslash, escape that character with a backslash. The backslash is the escape character in the AWP command compiler.

Table 11- 2 Examples of Name clauses

Data block name	Tag name	Name clause options
n/a	ABC'DEF	`Name='"ABC\'DEF"'`
n/a	A \B 'C :D	`Name='"A \\B \'C :D"'`
Data_block_1	Tag_1	`Name='"Data_block_1".Tag_1'`
Data_block_1	ABC'DEF	`Name='"Data_block_1".ABC\'DEF'`
Data_block_1	A \B 'C :D	`Name='"Data_block_1".A \\B \'C :D'`
DB A' B C D$ E	Tag	`Name='"DB A\' B C D$ E".Tag'`

Use clauses follow the same conventions as Name clauses.

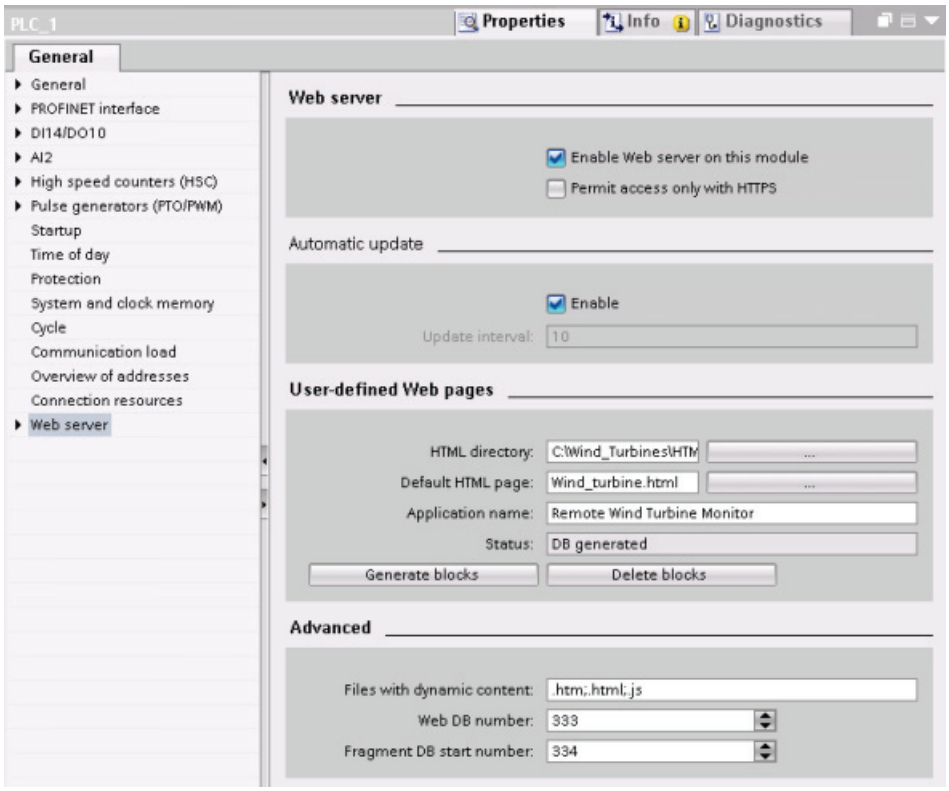
Note

Regardless of what characters you use in your HTML page, set the charset of the HTML page to UTF-8 and save it from the editor with UTF-8 character encoding.

11.3.3 Configuring use of user-defined Web pages

To configure user-defined Web pages from STEP 7, follow these steps:

1. Select the CPU in the Device Configuration view.
2. Display the "Web server" properties in the inspector window for the CPU.
3. If not already selected, select the check box for "Enable Web server".
4. Display the "User-defined Web pages" properties:



5. Enter or browse to the folder name on your PC where you saved the HTML default page (start page).
6. Enter the name of the default page.
7. Provide a name for your application (optional). The application name is used to further subcategorize or group web pages. When an application name exists, the URL will appear in the following format: http://ww.xx.yy.zz/awp/<application name>/<pagename>.html.
8. Specify filename extensions to be examined for the presence of AWP commands. By default, STEP 7 analyzes files with .htm, .html, or .js extensions. If you have additional file extensions, append them.
9. Keep the default for the Web DB number, or enter a number of your choice. This is the DB number of the control DB that controls display of the Web pages.
10. Keep the default for the fragment DB start number, or enter a number of your choice. This is the first of the fragment DBs that contains the Web pages.

Generating program blocks

When you click the "Generate blocks" button, STEP 7 generates data blocks from the HTML pages in the HTML source directory that you specified and a control data block for the operation of your Web pages. You can set these attributes as needed for your application (Page 508). STEP 7 also generates a set of fragment data blocks to hold the representation of all of your HTML pages. When you generate the data blocks, STEP 7 updates the properties to display the control data block number, and the number of the first of the fragment data blocks. After you generate the data blocks, your user-defined Web pages are a part of your STEP 7 program. The blocks corresponding to these pages appear in the Web server folder, which is in the System blocks folder under Program blocks in the project navigation tree.

Deleting program blocks

To delete data blocks that you have previously generated, click the "Delete data blocks" button. STEP 7 deletes the control data block and all of the fragment data blocks from your project that correspond to user-defined Web pages.

11.3.4 Programming the WWW instruction for user-defined web pages

Your STEP 7 user program must include and execute the WWW instruction in order for the user-defined Web pages to be accessible from the standard Web pages. The control data block is the input parameter to the WWW instruction and specifies the content of the pages as represented in the fragment data blocks, as well as state and control information. STEP 7 creates the control data block when you click the "Create blocks" button in the configuration of user-defined Web pages (Page 507).

Programming the WWW instruction

The STEP 7 program must execute the WWW instruction for the user-defined Web pages to be accessible from the standard Web pages. You might want the user-defined Web pages available only under certain circumstances as dictated by your application requirements and preferences. In this case, your program logic can control when to call the WWW instruction.

Table 11- 3 WWW instruction

LAD / FBD	SCL	Description
	`ret_val := WWW(` `  ctrl_db:=_uint_in_);`	Provides access to user-defined Web pages from standard Web pages

You must provide the control data block input parameter (CTRL_DB) which corresponds to the integer DB number of the control DB. You can find this control DB block number (called Web DB Number) in the Web Server properties of the CPU after you create the blocks for the user-defined Web pages. Enter the integer DB number as the CTRL_DB parameter of the WWW instruction. The return value (RET_VAL) contains the function result. Note that the WWW instruction executes asynchronously and that the RET_VAL output might have an initial value of 0 although an error can occur later. The program can check the state of the control DB to ensure that the application started successfully, or check RET_VAL with a subsequent call to WWW.

Table 11- 4 Return value

RET_VAL	Description
0	No error
16#00yx	x: The request represented by the respective bit is in the waiting state: x=1: request 0 x=2: request 1 x=4: request 2 x=8: request 3 The x values can be logically OR-ed to represent waiting states of multiple requests. If x = 6, for example, requests 1 and 2 are waiting. y: 0: no error; 1: error exists and "last_error" has been set in the control DB
16#803a	The control DB is not loaded.
16#8081	The control DB is of the wrong type, format, or version.
16#80C1	No resources are available to initialize the web application.

Usage of the Control DB

STEP 7 creates the control data block when you click "Generate blocks" and displays the control DB number in the User-defined Web pages properties. You can find the control DB as well in the Program blocks folder in the project navigation tree.

Typically, your STEP 7 program uses the control DB directly as created by the "Generate blocks" process with no additional manipulation. However, the STEP 7 user program can set global commands in the control DB to deactivate the web server or to subsequently re-enable it. Also, for user-defined pages that you create as manual fragment DBs (Page 507), the STEP 7 user program must control the behavior of these pages through a request table in the control DB. For information on these advanced tasks, see the topic Advanced user-defined Web page control (Page 526).

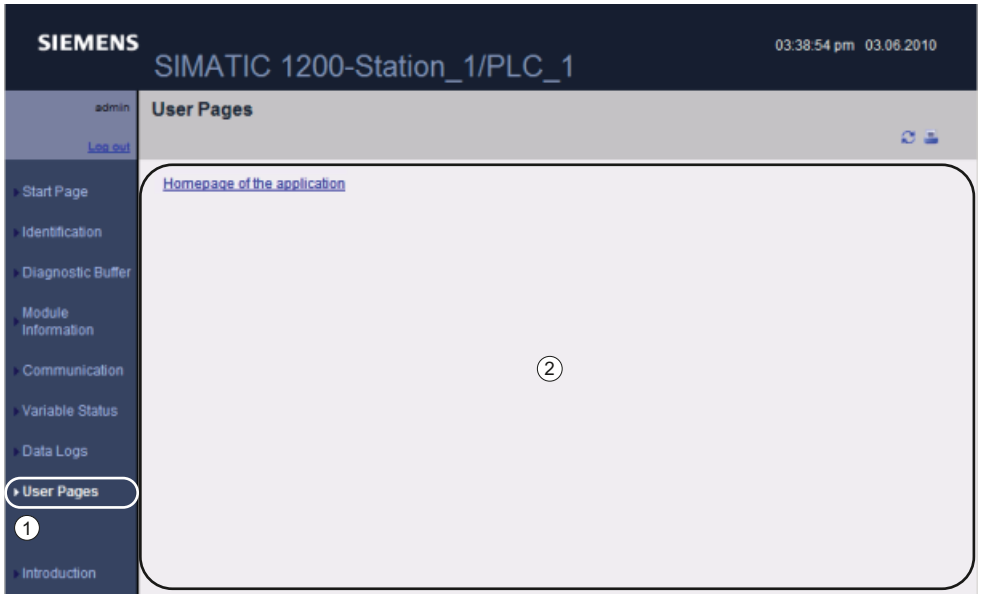
11.3.5 Downloading the program blocks to the CPU

After you have generated the blocks for user-defined Web pages, they are part of your STEP 7 program just like any other program blocks. You follow the normal process to download the program blocks to the CPU. Note that you can only download user-defined Web page program blocks when the CPU is in STOP mode.

11.3.6 Accessing the user-defined web pages from the PC

You access your user-defined Web pages from the standard Web pages (Page 474). The standard Web pages display a link for "User Pages" on the left side menu where the links to the other pages appear. When you click the "User Pages" link, your Web browser goes to the page that provides access to your default page. From within the user-defined content area, navigation is according to how you designed your specific pages.

The Web server displays the user-defined page content within the framework of the standard Web page display. That is, the header frame at the top and the login and navigation frames at the left remain in fixed positions.



① Link to default page of user-defined Web pages
② Content area of user-defined Web pages

11.3.7 Constraints specific to user-defined Web pages

The constraints for standard Web pages (Page 488) also apply to user-defined Web pages. In addition, user-defined Web pages have some specific considerations.

Load memory space

Your user-defined Web pages become data blocks when you click "Generate blocks", which require load memory space. If you have a memory card installed, you have up to the capacity of your memory card as external load memory space for the user-defined Web pages.

If you do not have a memory card installed, these blocks take up internal load memory space, which is limited according to your CPU model.

You can check the amount of load memory space that is used and the amount that is available from the Online and Diagnostic tools in STEP 7. You can also look at the properties for the individual blocks that STEP 7 generates from your user-defined Web pages and see the load memory consumption.

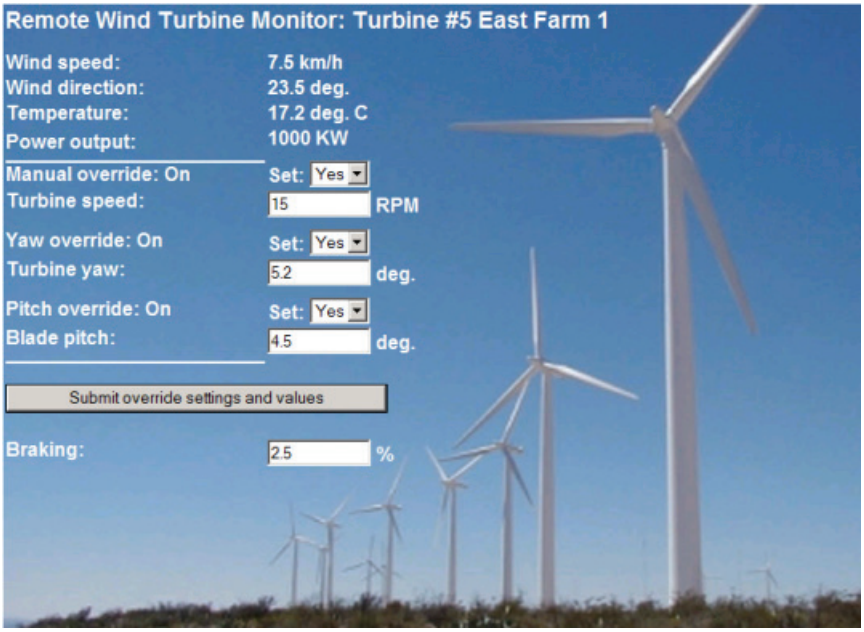
Note

If you need to reduce the space required for your user-defined Web pages, reduce your use of images if applicable.

11.3.8 Example of a user-defined web page

11.3.8.1 Web page for monitoring and controlling a wind turbine

As an example of a user-defined Web page, consider a Web page that is used to remotely monitor and control a wind turbine:



Note

This example page is in English, but you can, of course, use any language when you are developing your own HTML pages.

Description

In this application, each wind turbine in a wind turbine farm is equipped with an S7-1200 for control of the turbine. Within the STEP 7 program, each wind turbine has a data block with data specific to that wind turbine and location.

The user-defined Web page provides remote turbine access from a PC. A user can connect to standard web pages of the CPU of a particular wind turbine and access the user-defined "Remote Wind Turbine Monitor" Web page to see the data for that turbine. An admin user can also put the turbine in manual mode and control the variables for turbine speed, yaw, and pitch from the Web page. An admin user can also set a braking value regardless of whether the turbine is under manual or automatic control.

The STEP 7 program would check the Boolean values for overriding automatic control, and if set, would use the user-entered values for turbine speed, yaw, and pitch. Otherwise, the program would ignore these values.

Files used

This user-defined Web page example consists of three files:

- **Wind_turbine.html**: This is the HTML page that implements the display shown above, using AWP commands to access controller data.
- **Wind_turbine.css**: This is the cascading style sheet that contains formatting styles for the HTML page. Use of a cascading style sheet is optional, but it can simplify the HTML page development.
- **Wind_turbine.jpg**: This is the background image that the HTML page uses. Use of images in user-defined Web pages is, of course, optional, and does require additional space in the CPU.

These files are not provided with your installation, but are described as an example.

Implementation

The HTML page uses AWP commands to read values from the PLC (Page 495) for the display fields and to write values to the PLC (Page 496) for data coming from user input. This page also uses AWP commands for enum type definition (Page 501) and reference (Page 501) for handling ON/OFF settings.

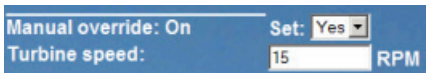
The first part of the page displays a header line that includes the wind turbine number and the location.

The next part of the page displays atmospheric conditions at the wind turbine. These fields are supplied from I/O at the turbine site that provide the wind speed, wind direction, and current temperature.

Next, the page displays the power output of the turbine as read from the S7-1200.

The following sections allow for manual control of the turbine, overriding the normal automatic control by the S7-1200. These types are as follows:

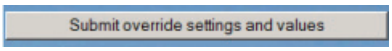
- Manual override: enables manual override of the turbine. The STEP 7 user program requires that the manual override setting be true before enabling the use of any of the manual settings for turbine speed, or yaw or pitch.



- Yaw override: enables manual override of the yaw setting, and a manual setting for the yaw. The STEP 7 user program requires that both manual override and yaw override be true in order to apply the yaw setting.

- Pitch override: enables manual override of the pitch of the blades. The STEP 7 user program requires that both manual override and pitch override be true in order to apply the blade pitch setting.

The HTML page includes a submit button to post the override settings to the controller.



The braking user input field provides a manual setting for a braking percentage. The STEP 7 user program does not require manual override to accept a braking value.

In addition, the HTML page uses an AWP command to write the special variable (Page 499) that contains the user ID of the user that is accessing the page to a tag in the PLC tag table.

11.3.8.2 Reading and displaying controller data

The "Remote Wind Turbine Monitor" HTML page uses numerous AWP commands for reading data from the controller (Page 495) and displaying it on the page. For example, consider the HTML code for displaying the power output as shown in this portion of the example Web page:

Sample HTML code

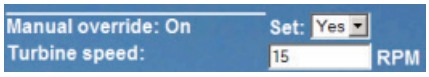
The following excerpt from the "Remote Wind Turbine Monitor" HTML page displays the text "Power Output:" in the left cell of a table row and reads the variable for the power output and displays it in the right cell of the table row along with the text for the units, KW.

The AWP command :="Data_block_1".PowerOutput: performs the read operation. Note that data blocks are referenced by name, not by data block number (that is, "Data_block_1" and not "DB1").

```
<tr style="height:2%;">
<td>
<p>Power output:</p>
</td>
<td>
<p style="margin-bottom:5px;"> :="Data_block_1".PowerOutput: KW</p>
</td>
</tr>
```

11.3.8.3 Using an enum type

%The "Remote Wind Turbine Monitor" HTML page uses enum types for the three instances where HTML page displays "ON" or "OFF" for a Boolean value, and for where the user sets a Boolean value. The enum type for "ON" results in a value of 1, and the enum type for "OFF" results in a value of 0. For example, consider the HTML code for reading and writing the Manual Override Enable setting in "Data_block_1".ManualOverrideEnable value using an enum type:



Sample HTML code

The following excerpts from the "Remote Wind Turbine Monitor" HTML page show how to declare an enum type called "OverrideStatus" with values of "Off" and "On" for 0 and 1, and then setting an enum type reference to OverrideStatus for the ManualOverrideEnable Boolean tag in the data block named "Data_block_1". Note that for each AWP_Enum_Ref declaration there must be a corresponding AWP_In_Variable declaration for the data block tag or PLC variable if the HTML page writes to the variable through an enum type.

```
<!-- AWP_In_Variable Name='"Data_block_1".ManualOverrideEnable' -->

<!-- AWP_Enum_Def Name="OverrideStatus" Values='0:"Off",1:"On"' -->

<!-- AWP_Enum_Ref Name='"Data_block_1".ManualOverrideEnable'
Enum="OverrideStatus" -->
```

Where the HTML page includes a display field in a table cell for the current state of ManualOverrideEnable. It uses just a normal read variable command, but with the use of the previously declared and referenced enum type, the page displays "Off" or "On" rather than 0 or 1.

```
<td style="width:24%; border-top-style: Solid; border-top-width:
2px; border-top-color: #ffffff;">
<p>Manual override: :="Data_block_1".ManualOverrideEnable:</p>
</td>
```

The HTML page includes a drop-down select list for the user to change the value of ManualOverrideEnable. The select list uses the text "Yes" and "No" to display in the select lists. With the use of the enum type, "Yes" is correlated to the value "On" of the enum type, and "No" is correlated to the value "Off". The empty selection leaves the value of ManualOverrideEnable as it is.

```
<select name='"Data_block_1".ManualOverrideEnable'>
<option value=':"Data_block_1".ManualOverrideEnable:'> </option>
<option value="On">Yes</option>
<option selected value="Off">No</option>
</select>
```

The select list is included within a form on the HTML page. When the user clicks the submit button, the page posts the form, which writes a value of "1" to the Boolean ManualOverrideEnable in Data_block_1 if the user had selected "Yes", or "0" if the user had selected "No".

11.3.8.4 Writing user input to the controller

The "Remote Wind Turbine Monitor" HTML page includes several AWP commands for writing data to the controller (Page 496). The HTML page declares AWP_In_Variables for Boolean variables so that an admin user can put the wind turbine under manual control and enable manual override for the turbine speed, yaw override, and/or blade pitch override. The page also uses AWP_In_Variables to allow an admin user to subsequently set floating-point values for the turbine speed, yaw, pitch, and braking percentage. The page uses an HTTP form post command to write the AWP_In_Variables to the controller.

For example, consider the HTML code for manually setting the braking value:

Sample HTML code

The following excerpt from the "Remote Wind Turbine Monitor" HTML page first declares an AWP_In_Variable for "Data_block_1" that enables the HTML page to write to any tags in the data block "Data_block_1". The page displays the text "Braking:" in the left cell of a table row. In the right cell of the table row is the field that accepts user input for the "Braking" tag of "Data_block_1". This user input value is within an HTML form that uses the HTTP method "POST" to post the entered text data to the CPU. The page then reads the actual braking value from the controller and displays it in the data entry field.

An admin user can subsequently use this page to write a braking value to the data block in the CPU that controls braking.

```
<!-- AWP_In_Variable Name='"Data_block_1"' -->
...
<tr style="vertical-align: top; height: 2%;">
<td style="width: 22%;"><p>Braking:</p></td>
<td>
<form method="POST">
<p><input name='"Data_block_1".Braking' size="10" type="text"> %</p>
</form>
</td>
</tr>
```

Note

Note that if a user-defined page has a data entry field for a writable data block tag that is a string data type, the user must enclose the string in single quotation marks when entering the string value in the field.

Note

Note that if you declare an entire data block in an AWP_In_Variable declaration such as <!-- AWP_In_Variable Name="'Data_block_1"' -->, then every tag within that data block can be written from the user-defined Web page. Use this when you intend for all of the tags in a data block to be writable. Otherwise, if you only want specific data block tags to be writable from the user-defined Web page, declare it specifically with a declaration such as <!-- AWP_In_Variable Name="'Data_block_1".Braking' -->

11.3.8.5 Writing a special variable

The "Remote Wind Turbine Monitor" Web page writes the special variable SERVER:current_user_id to a PLC tag in the CPU. In this case, the PLC tag value contains the user ID of whoever is accessing the "Remote Wind Turbine Monitor" Web page. Currently the admin user has a user ID of 1, so the PLC tag value is set to 1.

The special variable is written to the PLC by the Web page and requires no user interface.

Sample HTML code

```
<!-- AWP_In_Variable Name="SERVER:current_user_id" Use="User_ID"-->
```

11.3.8.6 Reference: HTML listing of remote wind turbine monitor Web page

Wind_turbine.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd"><!--
This test program simulates a Web page to monitor and control a Wind
Turbine
Required PLC tags and Data Block Tags in STEP 7:

PLC Tag:
User_ID: Int

Data Blocks:
Data_block_1

Tags in Data_Block_1:
```

```
Location: String
TurbineNumber: Int
WindSpeed: Real
WindDirection: Real
Temperature: Real
PowerOutput: Real
ManualOverrideEnable: Bool
TurbineSpeed: Real
YawOverride: Bool
Yaw: Real
PitchOverride: Bool
Pitch: Real
Braking: Real

The user-defined Web page displays current values for the PLC data,
and provides a select list to set the three booleans using an
enumerated type assignment. The "Submit" button posts the selected
boolean values as well as the data entry fields for TurbineSpeed,
Yaw, and Pitch. The value for Braking can be set without use of the
"Submit" button.

No actual STEP 7 program is required to use this page.
Theoretically, the STEP 7 program would only act on the values of
TurbineSpeed, Yaw, and Pitch, if the associated booleans were set.
The only STEP 7 requirement is to call the WWW instruction with the
DB number of the generated data blocks for this page.
-->
<!-- AWP_In_Variable Name='"Data_block_1"' -->
<!-- AWP_In_Variable Name='"Data_block_1".ManualOverrideEnable' -->
<!-- AWP_In_Variable Name='"Data_block_1".PitchOverride' -->
<!-- AWP_In_Variable Name='"Data_block_1".YawOverride' -->
<!-- AWP_In_Variable Name="SERVER:current_user_id" Use="User_ID"-->
<!-- AWP_Enum_Def Name="OverrideStatus" Values='0:"Off",1:"On"' -->
<!-- AWP_Enum_Ref Name='"Data_block_1".ManualOverrideEnable'
Enum="OverrideStatus" -->
<!-- AWP_Enum_Ref Name='"Data_block_1".PitchOverride'
Enum="OverrideStatus" -->
<!-- AWP_Enum_Ref Name='"Data_block_1".YawOverride'
Enum="OverrideStatus" -->

<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-
8"><link rel="stylesheet" href="Wind_turbine.css">
<title>Remote Wind Turbine Monitor</title>
</head>
<body>
<table style="background-image: url('./Wind_turbine.jpg'); width:
904px; height: 534px;" cellpadding="0" cellspacing="2"><tr
style="height: 2%;"><td colspan="2">
<h2>Remote Wind Turbine Monitor: Turbine
#:="Data_block_1".TurbineNumber: :="Data_block_1".Location:</h2>
</td>
```

```
<tr style="height: 2%;"><td style="width: 24%;"><p>Wind
speed:</p></td>
<td><p> :="Data_block_1".WindSpeed: km/h</p></td>
</tr>

<tr style="height: 2%;">
<td style="width: 24%;"><p>Wind direction:</p></td>
<td><p> :="Data_block_1".WindDirection: deg.</p></td>
</tr>

<tr style="height: 2%;"><td style="width:
24%;"><p>Temperature:</p></td>
<td><p> :="Data_block_1".Temperature: deg. C</p></td>
</tr>

<tr style="height: 2%;">
<td style="width: 24%;"><p>Power output:</p></td>
<td><p style="margin-bottom:5px;"> :="Data_block_1".PowerOutput:
KW</p>
</td>
</tr>

<form method="POST" action="">
<tr style="height: 2%;" >
<td style="width=24%; border-top-style: Solid; border-top-width:
2px; border-top-color: #ffffff;">
<p>Manual override: :="Data_block_1".ManualOverrideEnable:</p>
</td>
<td class="Text">Set:

<select name='"Data_block_1".ManualOverrideEnable'>
<option value=':="Data_block_1".ManualOverrideEnable:'> </option>
<option value="On">Yes</option>
<option value="Off">No</option>
</select>

</td>
</tr>

<tr style="vertical-align: top; height: 2%;"><td style="width:
24%;"><p>Turbine speed:</p></td>
<td>
<p style="margin-bottom:5px;"><input
name='"Data_block_1".TurbineSpeed' size="10"
value=':="Data_block_1".TurbineSpeed:' type="text"> RPM</p>
</td>
</tr>

<tr style="vertical-align: top; height: 2%;">
<td style="width: 24%;">
<p>Yaw override: :="Data_block_1".YawOverride: </p>
</td>
<td class="Text">Set:

<select name='"Data_block_1".YawOverride'>
```

```
<option value=':="Data_block_1".YawOverride:'> </option>
<option value="On">Yes</option>
<option value="Off">No</option>
</select>

</td>
</tr>

<tr style="vertical-align: top; height: 2%;">
<td style="width: 24%;">
<p>Turbine yaw:</p>
</td>
<td>
<p style="margin-bottom:5px;"><input name='"Data_block_1".Yaw'
size="10" value=':="Data_block_1".Yaw:' type="text"> deg.</p>
</td>
</tr>

<tr style="vertical-align: top; height: 2%;">
<td style="width: 24%;">
<p>Pitch override: :="Data_block_1".PitchOverride: </p>
</td>
<td class="Text">Set:

<select name='"Data_block_1".PitchOverride'>
<option value=':="Data_block_1".PitchOverride:'> </option>
<option value="On">Yes</option>
<option value="Off">No</option>
</select>

</td>
</tr>

<tr style="vertical-align: top; height: 2%;">
<td style="width=24%; border-bottom-style: Solid; border-bottom-
width: 2px; border-bottom-color: #ffffff;">
<p>Blade pitch:</p>
</td>
<td>
<p style="margin-bottom:5px;"><input name='"Data_block_1".Pitch'
size="10" value=':="Data_block_1".Pitch:' type="text"> deg.</p>
</td>

</tr>
<tr style="height: 2%;">
<td colspan="2"><br>
<input type="submit" value="Submit override settings and values">
</td>
</tr>
</form>

<tr style="vertical-align: top; height: 2%;">
<td style="width: 24%;"><p>Braking:</p></td>
<td>
<form method="POST" action="">
```

```
<p> <input name='"Data_block_1".Braking' size="10"
value=':="Data_block_1".Braking:' type="text"> %</p>
</form>
</td>
</tr>
<tr><td></td></tr>

</table>
</body>
</html>
```

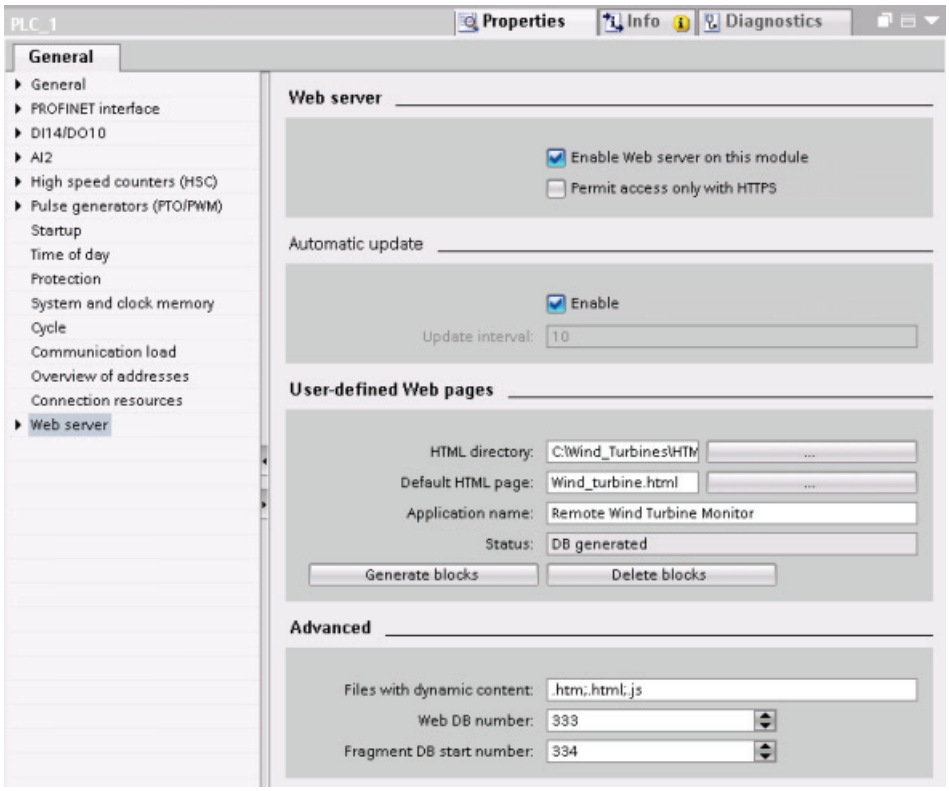
Wind_turbine.css

```
H2 {
 font-family: Arial;
 font-weight: bold;
 font-size: 14.0pt;
 color: #FFFFFF;
 margin-top:0px;
 margin-bottom:10px;
}
P {
 font-family: Arial;
 font-weight: bold;
 color: #FFFFFF;
 font-size: 12.0pt;
 margin-top:0px;
 margin-bottom:0px;
}
TD.Text {
 font-family: Arial;
 font-weight: bold;
 color: #FFFFFF;
 font-size: 12.0pt;
 margin-top:0px;
 margin-bottom:0px;
}
```

11.3.8.7 Configuration in STEP 7 of the example Web page

To include the "Remote Wind Turbine Monitor" HTML page as a user-defined Web page for the S7-1200, you configure the data about the HTML page in STEP 7 and create data blocks from the HTML page.

Access the CPU Properties for the S7-1200 that controls the wind turbine, and enter the configuration information in the User-defined web pages properties of the Web Server:



Configuration fields

- HTML directory: This field specifies the fully-qualified pathname to the folder where the default page (home page or start page) is located on the computer. The "..." button allows you to browse to the folder that you need.
- Default HTML page: This field specifies the filename of the default page or home page of the HTML application. The "..." button allows you to select the file that you need. For this example, WindTurbine.html is the default HTML page. The Remote Wind Turbine Monitor example only consists of a single page, but in other user-defined applications the default page can call up additional pages from links on the default page. Within the HTML code, the default page must reference other pages relative to the HTML source folder.
- Application name: This optional field contains the name that the Web browser includes in the address field when displaying the page. For this example, it is "Remote Wind Turbine Monitor", but it can be any name.

No other fields require configuration.

Final steps

To use the Remote Wind Turbine Monitor as configured, generate the blocks, program the WWW instruction (Page 508) with the number of the generated control DB as an input parameter, download the program blocks, and put the CPU in run mode.

When an operator subsequently accesses the standard Web pages for the S7-1200 that controls the wind turbine, the "Remote Wind Turbine Monitor" Web page is accessible from the "User Pages" link on the navigation bar. This page now provides the means to monitor and control the wind turbine.

11.3.9 Setting up user-defined Web pages in multiple languages

The Web server provides the means for you to provide user-defined Web pages in the following languages:

- German (de)
- English (en)
- Spanish (es)
- French (fr)
- Italian (it)
- Chinese (zh)

You do this by setting up your HTML pages in a folder structure (Page 522) that corresponds to the languages and by setting a specific cookie named "siemens_automation_language" from your pages (Page 523). The Web server responds to this cookie, and switches to the default page in the corresponding language folder.

11.3.9.1 Creating the folder structure

To provide user-defined Web pages in multiple languages, you set up a folder structure under your HTML directory. The two-letter folder names are specific and must be named as shown below:

de: German
en: English
es: Spanish
fr: French
it: Italian
zh: Chinese

At the same level, you can also include any other folders that your pages need, for example, folders for images or scripts.

You can include any subset of the language folders. You do not have to include all six languages. Within the language folders, you create and program your HTML pages in the appropriate language.

11.3.9.2 Programming the language switch

The Web server performs switching between languages through the use of a cookie named "siemens_automation_language". This is a cookie defined and set in the HTML pages, and interpreted by the Web server to display a page in the appropriate language from the language folder of the same name. The HTML page must include a JavaScript to set this cookie to one of the pre-defined language identifiers: "de", "en", "es", "fr", "it", or "zh".

For example, if the HTML page sets the cookie to "de", the Web server switches to the "de" folder and displays the page with the default HTML page name as defined in the STEP 7 configuration (Page 525).

Example

The following example uses a default HTML page named "langswitch.html" in each of the language folders. Also in the HTML directory is a folder named "script". The script folder includes a JavaScript file named "lang.js". Each langswitch.html page uses this JavaScript to set the language cookie, "siemens_automation_language".

HTML for "langswitch.html" in "en" folder

The header of the HTML page sets the language to English, sets the character set to UTF-8, and sets the path to the JavaScript file lang.js.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Language" content="en">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Language switching english page</title>
<script type="text/javascript" src="script/lang.js" ></script>
```

The body of the file uses a select list for the user to select between German and English. English ("en") is pre-selected for the language. When the user changes the language, the page calls the DoLocalLanguageChange() JavaScript function with the value of the selected option.

```
<!-- Language Selection -->
<table>
   <tr>
      <td align="right" valign="top" nowrap>
       <!-- change language immediately on selection change -->
          <select name="Language"
                    onchange="DoLocalLanguageChange(this)"
                    size="1">
            <option value="de" >German</option>
            <option value="en" selected >English</option>
           </select>
      </td>
   </tr>
</table><!-- Language Selection End-->
```

HTML for "langswitch.html" in "de" folder

The header for the German langswitch.html page is the same as English, except the language is set to German.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Language" content="de"><meta http-
equiv="Content-Type" content="text/html; charset=utf-8">
<title>Sprachumschaltung Deutsche Seite</title>
<script type="text/javascript" src="script/lang.js" ></script>
</head>
```

The HTML in the German page is identical to that of the English page, except that the default value of the selected language is German ("de").

```
<!-- Language Selection -->
<table>
   <tr>
     <td align="right" valign="top" nowrap>
       <!-- change language immediately on change of the selection -
->
       <select name="Language"
               onchange="DoLocalLanguageChange(this)"
               <size="1">
            <option value="de" selected >Deutsch</option>
            <option value="en" >Englisch</option>
       </select>
      </td>
   </tr>
</table><!-- Language Selection End-->
```

JavaScript "lang.js" in "script" folder

The function "DoLocalLanguageChange()" is in the lang.js file. This function calls the "SetLangCookie()" function and then reloads the window that is displaying the HTML page.

The function "SetLangCookie()" constructs an assignment that assigns the value from the select list to the "siemens_automation_language" cookie of the document. It also sets the path to the application so that the switched page, and not the requesting page, receives the value of the cookie.

Optionally, in the commented section, the page could set an expiration value for the cookie.

```
function DoLocalLanguageChange(oSelect) {
        SetLangCookie(oSelect.value);
        top.window.location.reload();
    }
function SetLangCookie(value) {
        var strval = "siemens_automation_language=";
        // This is the cookie by which the Web server
        // detects the desired language
        // This name is required by the Web server.
        strval = strval + value;
        strval = strval + "; path=/ ;";
        // Set path to the application, since otherwise
        // path would be set to the requesting page
        // and this page would not get the cookie.
        /* OPTIONAL
           use expiration if this cookie should live longer
           than the current browser session:
           var now = new Date();
           var endttime = new Date(now.getTime() + expiration);
           strval = strval + "; expires=" +
                    endttime.toGMTString() + ";";
        */
        document.cookie = strval;
    }
```

11.3.9.3 Configuring STEP 7 to use a multi-language page structure

The procedure for configuring multi-language user-defined Web pages is similar to the general process for configuring user-defined Web pages (Page 507). When you have folders set up for languages, however, you set your HTML directory setting to the folder that contains the individual language folders. You do not set the HTML directory to be one of the language folders.

When you select the default HTML page, you navigate into the language folder and select the HTML page that is to be the start page. When you subsequently generate blocks and download the blocks to the CPU, the Web server displays the start page in the language folder that you configured.

For example, if the folder structure shown here was at C:\, the setting for HTML directory would be C:\html, and if English were to be the initial page display, you would navigate to en\langswitch.html for the default HTML page setting.

11.3.10 Advanced user-defined Web page control

When you generate data blocks for your user-defined Web pages, STEP 7 creates a control DB that it uses to control display of and interaction with the user-defined pages. STEP 7 also creates a set of fragment DBs that represent the individual pages. Under normal circumstances, you do not need to know the structure of the control DB or how to manipulate it.

If you want to turn a web application on and off, for example, or manipulate individual manual fragments, you use the control DB tags and the WWW instruction to do so.

Structure of the control DB

The control DB is an extensive data structure, and is accessible when programming your STEP 7 user program. Only some of the control data block tags are described here.

Commandstate structure

"Commandstate" is a structure that contains global commands and global states for the Web server.

Global commands in the "Commandstate" structure

The global commands apply to the Web server in general. You can deactivate the Web server or restart it from the control DB parameters.

Block tag	Data type	Description
init	BOOL	Evaluate the control DB and initialize the Web application
deactivate	BOOL	Deactivate the Web application

Global states in the Commandstate structure

The global states apply to the Web server in general and contain status information about the Web application.

Block tag	Data type	Description
initializing	BOOL	Web application is reading control DB
error	BOOL	Web application could not be initialized
deactivating	BOOL	Web application is terminating
deactivated	BOOL	Web application is terminated
initialized	BOOL	Web application is initialized

Request table

The request table is an array of structures containing commands and states that apply to individual fragment DBs. If you created fragments with the AWP_Start_Fragment (Page 503) command of type "manual", the STEP 7 user program must control these pages through the control DB. The request states are read-only and provide information about the current fragment. You use the request commands to control the current fragment.

Block tag	Data type	Description
requesttab	ARRAY [1 .. 4] OF STRUCT	Array of structures for individual fragment DB control. The Web server can process up to four fragments at a time. The array index for a particular fragment is arbitrary when the Web server is processing multiple fragments or fragments from multiple browser sessions.

Struct members of requesttab struct

Block tag	Data type	Description
page_index	UINT	Number of the current web page
fragment_index	UINT	Number of the current fragment - can be set to a different fragment
// Request Commands		
continue	BOOL	Enables current page/fragment for sending and continues with the next fragment
repeat	BOOL	Enables current page/fragment for resending and continues with the same fragment
abort	BOOL	Close http connection without sending
finish	BOOL	Send this fragment; page is complete - do not process any additional fragments
// Request states		The request states are read-only
idle	BOOL	Nothing to do, but active
waiting	BOOL	Fragment is waiting to be enabled
sending	BOOL	Fragment is sending
aborting	BOOL	User has aborted current request

Operation

Whenever your program makes changes to the control DB, it must call the WWW instruction with the number of the modified control DB as its parameter. The global commands and request commands take effect when the STEP 7 user program executes the WWW instruction (Page 508).

The STEP 7 user program can set the fragment_index explicitly, thus causing the Web server to process the specified fragment with a request command. Otherwise, the Web server processes the current fragment for the current page when the WWW instruction executes.

Possible techniques for using the fragment_index include:

- Processing the current fragment: Leave fragment_index unchanged and set the continue command.
- Skip the current fragment: Set fragment_index to 0 and set the continue command.
- Replace current fragment with a different fragment: Set the fragment_index to the new fragment ID and set the continue command.

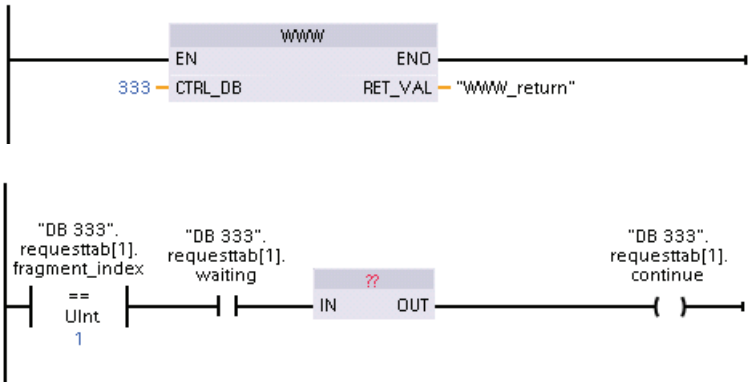
To check global states or request states that might be changing, the STEP 7 user program must call the WWW instruction to evaluate the current values of these states. A typical usage might be to call the WWW instruction periodically until a specific state occurs.

Note

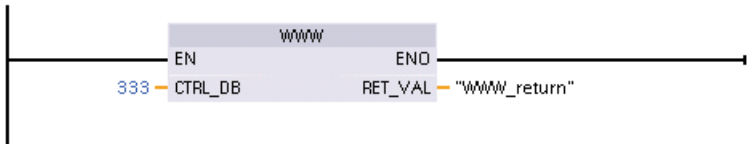
If the STEP 7 user program sets more than one request command, the WWW instructions processes only one in this of precedence: abort, finish, repeat, continue. The WWW instruction clears all of the request commands after processing.

Examples

The following example shows a STEP 7 user program that is checking for a fragment with an ID of 1 to be in the waiting state, following a prior call to the WWW instruction. It might also wait for other application-specific conditions to occur. Then it performs whatever processing is necessary for the fragment, such as setting data block tags, performing calculations, or other application-specific tasks. Afterwards, it sets the continue flag so that the Web server will execute this fragment.



When the program calls the WWW instruction with this modified control DB, the user-defined Web page with this fragment can be displayed from the Web browser.



Note that this is a simplified example; the fragment to check could be in any one of the four requesttab structs in the array.

Communication processor 12

12.1 Using the RS232 and RS485 communication interfaces

Three communication modules (CMs) and one communication board (CB) provide the interface for PtP communications:

- CM 1241 RS232 (Page 755)
- CM 1241 RS485 (Page 754)
- CM 1241 RS422/485 (Page 756)
- CB 1241 RS485 (Page 752)

You can connect up to three CMs (of any type) plus a CB for a total of four communication interfaces. Install the CM to the left of the CPU or another CM. Install the CB on the front of the CPU. Refer to the "Installation" chapter (Page 48) for detailed instructions on module installation and removal.

The RS232 and RS485 communication interfaces have the following characteristics:

- Have an isolated port
- Support Point-to-Point protocols
- Are configured and programmed through extended instructions and library functions
- Display transmit and receive activity by means of LEDs
- Display a diagnostic LED (CMs only)
- Are powered by the CPU: No external power connection is needed.

Refer to the technical specifications for communication interfaces (Page 743).

LED indicators

The communication modules have three LED indicators:

- Diagnostic LED (DIAG): This LED flashes red until it is addressed by the CPU. After the CPU powers up, it checks for a CB or CMs and addresses them. The diagnostic LED begins to flash green. This means that the CPU has addressed the CM or CB, but has not yet provided the configuration to it. The CPU downloads the configuration to the configured CMs and CB when the program is downloaded to the CPU. After a download to the CPU, the diagnostic LED on the communication module or communication board should be a steady green.
- Transmit LED (Tx): The transmit LED illuminates when data is being transmitted out the communication port.
- Receive LED (Rx): This LED illuminates when data is being received by the communication port.

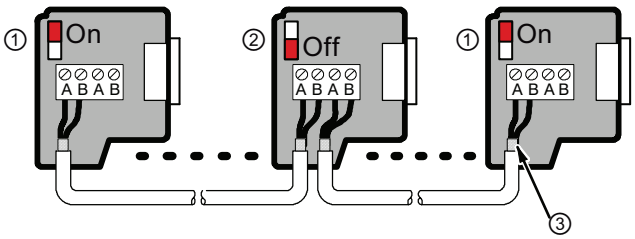
The communication board provides transmit (TxD) and receive (RxD) LEDs. It has no diagnostic LED.

12.2 Biasing and terminating an RS485 network connector

Siemens provides an RS485 network connector (Page 766) that you can use to easily connect multiple devices to an RS485 network. The connector has two sets of terminals that allow you to attach the incoming and outgoing network cables. The connector also includes switches for selectively biasing and terminating the network.

Note

You terminate and bias only the two ends of the RS485 network. The devices in between the two end devices are not terminated or biased.

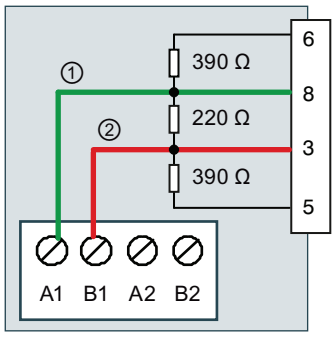
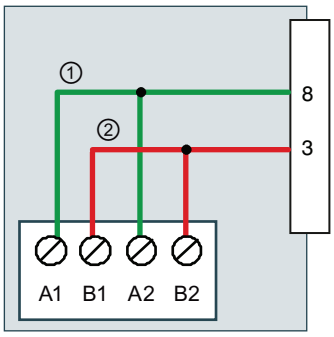


① End (terminating) devices: Switch position = ON (to terminate and bias the connector)

② Other (non-terminating) devices on the network: Switch position = OFF (no termination or bias for the connector)

③ Bare cable shielding: Approximately 12 mm must contact the metal guides of all locations.

Table 12- 1 Termination and bias for the RS485 connector

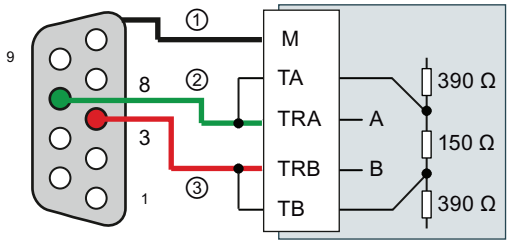
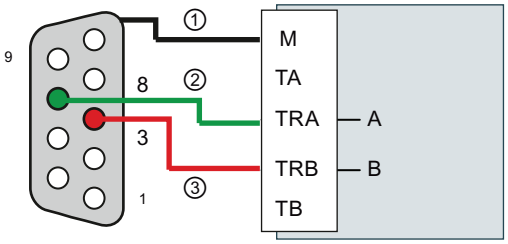
Terminating device (bias ON)	Non-terminating device (bias OFF)
	

① A = TxD/RxD - (Green wire / Pin 8)

② B = TxD/RxD + (Red wire / Pin 3)

The CB 1241 provides internal resistors for terminating and biasing the network. To terminate and bias the connection, connect TRA to TA and connect TRB to TB to include the internal resistors to the circuit. CB 1241 does not have a 9-pin connector. The following table shows the connections to a 9-pin connector on the communications partner.

Table 12- 2 Termination and bias for the CB 1241

Terminating device (bias ON)	Non-terminating device (bias OFF)
	

① Connect M to the cable shield

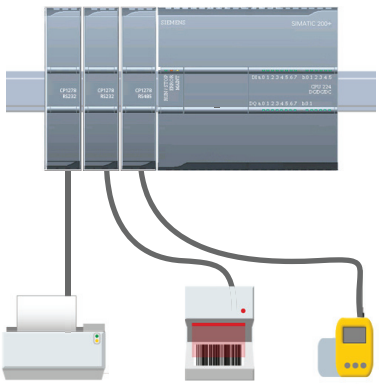
② A = TxD/RxD - (Green wire / Pin 8)

③ B = TxD/RxD + (Red wire / Pin 3)

12.3 Point-to-Point (PtP) communication

The CPU supports the following Point-to-Point communication (PtP) for character-based serial protocols. PtP provides maximum freedom and flexibility, but requires extensive implementation in the user program.

- PtP (Page 531)
- USS (Page 567)
- Modbus (Page 581)



PtP enables a wide variety of possibilities:

- The ability to send information directly to an external device such as a printer
- The ability to receive information from other devices such as barcode readers, RFID readers, third-party camera or vision systems, and many other types of devices
- The ability to exchange information, sending and receiving data, with other devices such as GPS devices, third-party camera or vision systems, radio modems, and many more

PtP communication is serial communication that uses standard UARTs to support a variety of baud rates and parity options. The RS232 and RS485 communication modules and the RS485 communication board provide the electrical interfaces for performing the PtP communications.

12.3.1 Point-to-Point instructions

12.3.1.1 Common parameters for Point-to-Point instructions

Table 12- 3 Common input parameters for the PTP instructions

Parameter	Description
REQ	Many of the PtP instructions use the REQ input to initiate the operation on a low to high transition. The REQ input must be high (TRUE) for one execution of an instruction, but the REQ input can remain TRUE for as long as desired. The instruction does not initiate another operation until it has been called with the REQ input FALSE so that the instruction can reset the history state of the REQ input. This is required so that the instruction can detect the low to high transition to initiate the next operation. When you place a PtP instruction in your program, STEP 7 prompts you to identify the instance DB. Use a unique DB for each PtP instruction call. This ensures that each instruction properly handles inputs such as REQ.
PORT	A port address is assigned during communication device configuration. After configuration, a default port symbolic name can be selected from the parameter assistant drop-list. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "Constants" tab of the PLC tag table.
Bit time resolution	Several parameters are specified in a number of bit times at the configured baud rate. Specifying the parameter in bit times allows the parameter to be independent of baud rate. All parameters that are in units of bit times can be specified to a maximum number of 65535. However, the maximum amount of time that a CM or CB can measure is eight seconds.

The DONE, NDR, ERROR, and STATUS output parameters of the PtP instructions provide execution completion status for the PtP operations.

Table 12- 4 DONE, NDR, ERROR, and STATUS output parameters

Parameter	Data type	Default	Description
DONE	Bool	FALSE	Set TRUE for one execution to indicate that the last request completed without errors; otherwise, FALSE.
NDR	Bool	FALSE	Set TRUE for one execution to indicate that the requested action has completed without error and that the new data has been received; otherwise, FALSE.
ERROR	Bool	FALSE	Set TRUE for one execution to indicate that the last request completed with errors, with the applicable error code in STATUS; otherwise, FALSE.
STATUS	Word	0	Result status: • If the DONE or NDR bit is set, then STATUS is set to 0 or to an informational code. • If the ERROR bit is set, then STATUS is set to an error code. • If none of the above bits are set, then the instruction returns status results that describe the current state of the function. STATUS retains its value for the duration of the execution of the function.

Note

The DONE, NDR, and ERROR parameters are set for one execution only. Your program logic must save temporary output state values in data latches, so you can detect state changes in subsequent program scans.

Table 12- 5 Common condition codes

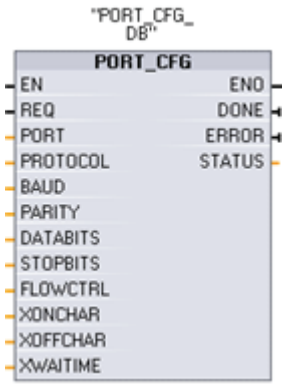
STATUS (W#16#....)	Description
0000	No error
7000	Function is not busy
7001	Function is busy with the first call.
7002	Function is busy with subsequent calls (polls after the first call).
8x3A	Illegal pointer in parameter x
8070	All internal instance memory in use, too many concurrent instructions in progress
8080	Port number is illegal.
8081	Timeout, module error, or other internal error
8082	Parameterization failed because parameterization is in progress in background.
8083	Buffer overflow: The CM or CB returned a received message with a length greater than the length parameter allowed.
8090	Internal error: Wrong message length, wrong sub-module, or illegal message Contact customer support.
8091	Internal error: Wrong version in parameterization message Contact customer support.
8092	Internal error: Wrong record length in parameterization message Contact customer support.

Table 12- 6 Common error classes

Class description	Error classes	Description
Port configuration	80Ax	Used to define common port configuration errors
Transmit configuration	80Bx	Used to define common transmit configuration errors
Receive configuration	80Cx	Used to define common receive configuration errors
Transmission runtime	80Dx	Used to define common transmission runtime errors
Reception runtime	80Ex	Used to define common reception runtime errors
Signal handling	80Fx	Used to define common errors associated with all signal handling

12.3.1.2 PORT_CFG instruction

Table 12- 7 PORT_CFG (Port Configuration) instruction

LAD / FBD	SCL	Description
	"PORT_CFG_DB"(REQ:=_bool_in_, PORT:=_uint_in_, PROTOCOL:=_uint_in_, BAUD:=_uint_in_, PARITY:=_uint_in_, DATABITS:=_uint_in_, STOPBITS:=_uint_in_, FLOWCTRL:=_uint_in_, XONCHAR:=_char_in_, XOFFCHAR:=_char_in_, WAITTIME:=_uint_in_, DONE=>_bool_out_, ERROR=>_bool_out_, STATUS=>_word_out_);	PORT_CFG allows you to change port parameters such as baud rate from your program. You can set up the initial static configuration of the port in the device configuration properties, or just use the default values. You can execute the PORT_CFG instruction in your program to change the configuration.

[1] STEP 7 automatically creates the DB when you insert the instruction.

The PORT_CFG configuration changes are not permanently stored in the CPU. The parameters configured in the device configuration are restored when the CPU transitions from RUN to STOP mode and after a power cycle. See Configuring the communication ports (Page 549) and Managing flow control (Page 550) for more information.

Table 12- 8 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Activate the configuration change on rising edge of this input. (Default value: False)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table. (Default value: 0)
PROTOCOL	IN	UInt	0 - Point-to-Point communication protocol (Default value) 1..n - future definition for specific protocols
BAUD	IN	UInt	Port baud rate (Default value: 0): 1 = 300 baud, 2 = 600 baud, 3 = 1200 baud, 4 = 2400 baud, 5 = 4800 baud, 6 = 9600 baud, 7 = 19200 baud, 8 = 38400 baud, 9 = 57600 baud, 10 = 76800 baud, 11 = 115200 baud
PARITY	IN	UInt	Port parity (Default value: 0): 1 = No parity, 2 = Even parity, 3 = Odd parity, 4 = Mark parity, 5 = Space parity
DATABITS	IN	UInt	Bits per character (Default value:): 1 = 8 data bits, 2 = 7 data bits
STOPBITS	IN	UInt	Stop bits (Default value: 0): 1 = 1 stop bit, 2 = 2 stop bits

Parameter and type		Data type	Description
FLOWCTRL	IN	UInt	Flow control (Default value: 0): 1 = No flow control, 2 = XON/XOFF, 3 = Hardware RTS always ON, 4 = Hardware RTS switched
XONCHAR	IN	Char	Specifies the character that is used as the XON character. This is typically a DC1 character (11H). This parameter is only evaluated if flow control is enabled. (Default value: 0)
XOFFCHAR	IN	Char	Specifies the character that is used as the XOFF character. This is typically a DC3 character (13H). This parameter is only evaluated if flow control is enabled. (Default value: 0)
XWAITIME	IN	UInt	Specifies how long to wait for a XON character after receiving a XOFF character, or how long to wait for the CTS signal after enabling RTS (0 to 65535 ms). This parameter is only evaluated if flow control is enabled. (Default value: 2000)
DONE	OUT	Bool	TRUE for one execution after the last request was completed with no error
ERROR	OUT	Bool	TRUE for one execution after the last request was completed with an error
STATUS	OUT	Word	Execution condition code (Default value: 0)

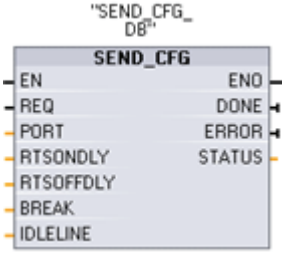
Table 12- 9 Condition codes

STATUS (W#16#....)	Description
80A0	Specific protocol does not exist.
80A1	Specific baud rate does not exist.
80A2	Specific parity option does not exist.
80A3	Specific number of data bits does not exist.
80A4	Specific number of stop bits does not exist.
80A5	Specific type of flow control does not exist.
80A6	Wait time is 0 and flow control enabled
80A7	XON and XOFF are illegal values (for example, the same value)

12.3.1.3 SEND_CFG instruction

Table 12- 10 SEND_CFG (Send Configuration) instruction

LAD / FBD	SCL	Description

	`"SEND_CFG_DB"(`
	`REQ:=_bool_in_,`
	`PORT:=_uint_in_,`
	`RTSONDLY:=_uint_in_,`
	`RTSOFFDLY:=_uint_in_,`
	`BREAK:=_uint_in_,`
	`IDLELINE:=_uint_in_,`
	`DONE=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_);`
		SEND_CFG allows the dynamic configuration of serial transmission parameters for a PtP communication port. Any queued messages within a CM or CB are discarded when SEND_CFG is executed.

[1] STEP 7 automatically creates the DB when you insert the instruction.

You can set up the initial static configuration of the port in the device configuration properties, or just use the default values. You can execute the SEND_CFG instruction in your program to change the configuration.

The SEND_CFG configuration changes are not permanently stored in the CPU. The parameters configured in the device configuration are restored when the CPU transitions from RUN to STOP mode and after a power cycle. See Configuring transmit (send) parameters (Page 552).

Table 12- 11 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Activate the configuration change on the rising edge of this input.. (Default value: False)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table. (Default value: 0)
RTSONDLY	IN	UInt	Number of milliseconds to wait after enabling RTS before any Tx data transmission occurs. This parameter is only valid when hardware flow control is enabled. The valid range is 0 - 65535 ms. A value of 0 disables the feature. (Default value: 0)
RTSOFFDLY	IN	UInt	Number of milliseconds to wait after the Tx data transmission occurs before RTS is disabled: This parameter is only valid when hardware flow control is enabled. The valid range is 0 - 65535 ms. A value of 0 disables the feature. (Default value: 0)
BREAK	IN	UInt	This parameter specifies that a break will be sent upon the start of each message for the specified number of bit times. The maximum is 65535 bit times up to an eight second maximum. A value of 0 disables the feature. (Default value: 12)
IDLELINE	IN	UInt	This parameter specifies that the line will remain idle for the specified number of bit times before the start of each message. The maximum is 65535 bit times up to an eight second maximum. A value of 0 disables the feature. (Default value: 12)
DONE	OUT	Bool	TRUE for one execution after the last request was completed with no error
ERROR	OUT	Bool	TRUE for one execution after the last request was completed with an error
STATUS	OUT	Word	Execution condition code (Default value: 0)

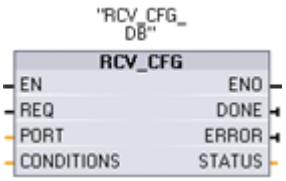
Table 12- 12 Condition codes

STATUS (W#16#....)	Description
80B0	Transmit interrupt configuration is not allowed. Contact customer suport.
80B1	Break time is greater than the maximum allowed value.
80B2	Idle time is greater than the maximum allowed value.

12.3.1.4 RCV_CFG instruction

Table 12- 13 RCV_CFG (Receive Configuration) instruction

LAD / FBD	SCL	Description

	`"RCV_CFG_DB"(`
	`REQ:=_bool_in_,`
	`PORT:=_uint_in_,`
	`CONDITIONS:=_struct_in_,`
	`DONE=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_);`
		RCV_CFG performs dynamic configuration of serial receiver parameters for a PtP communication port. This instruction configures the conditions that signal the start and end of a received message. Any queued messages within a CM or CB are discarded when RCV_CFG is executed.

[1] STEP 7 automatically creates the DB when you insert the instruction.

You can set up the initial static configuration of the communication port in the device configuration properties, or just use the default values. You can execute the RCV_CFG instruction in your program to change the configuration.

The RCV_CFG configuration changes are not permanently stored in the CPU. The parameters configured in the device configuration are restored when the CPU transitions from RUN to STOP mode and after a power cycle. See Configuring receive parameters (Page 552) for more information.

Table 12- 14 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Activate the configuration change on the rising edge of this input. (Default value: False)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table. (Default value: 0)
CONDITIONS	IN	CONDITIONS	The Conditions data structure specifies the starting and ending message conditions as described below.
DONE	OUT	Bool	TRUE for one scan, after the last request was completed with no error
ERROR	OUT	Bool	TRUE for one scan, after the last request was completed with an error
STATUS	OUT	Word	Execution condition code (Default value: 0)

Start conditions for the RCV_PTP instruction

The RCV_PTP instruction uses the configuration specified by the RCV_CFG instruction to determine the beginning and ending of point-to-point communication messages. The start of a message is determined by the start conditions. The start of a message can be determined by one or a combination of start conditions. If more than one start condition is specified, all the conditions must be satisfied before the message is started.

See the topic "Configuring receive parameters (Page 553)" for a description of the message start conditions.

Parameter CONDITIONS data type structure part 1 (start conditions)

Table 12- 15 CONDITIONS structure for start conditions

Parameter and type		Data type	Description
STARTCOND	IN	UInt	Specifies the start condition (Default value: 1) • 01H - Start Char • 02H - Any Char • 04H - Line Break • 08H - Idle Line • 10H - Sequence 1 • 20H - Sequence 2 • 40H - Sequence 3 • 80H - Sequence 4
IDLETIME	IN	UInt	The number of bit times required for idle line timeout. (Default value: 40). Only used with an idle line condition. 0 to 65535
STARTCHAR	IN	Byte	The start character used with the start character condition. (Default value: B#16#2)
STRSEQ1CTL	IN	Byte	Sequence 1 ignore/compare control for each character: (Default value: B#16#0) These are the enabling bits for each character in start sequence • 01H - Character 1 • 02H - Character 2 • 04H - Character 3 • 08H - Character 4 • 10H - Character 5 Disabling the bit associated with a character means any character will match, in this sequence position.
STRSEQ1	IN	Char[5]	Sequence 1 start characters (5 characters). Default value: 0
STRSEQ2CTL	IN	Byte	Sequence 2 ignore/compare control for each character. Default value: B#16#0)
STRSEQ2	IN	Char[5]	Sequence 2 start characters (5 characters). Default value: 0
STRSEQ3CTL	IN	Byte	Sequence 3 ignore/compare control for each character. Default value: B#16#0
STRSEQ3	IN	Char[5]	Sequence 3 start characters (5 characters). Default value: 0
STRSEQ4CTL	IN	Byte	Sequence 4 ignore/compare control for each character. Default value: B#16#0
STRSEQ4	IN	Char[5]	Sequence 4 start characters (5 characters), Default value: 0

Example

Consider the following received hexadecimal coded message: "**68** 10 aa **68** bb 10 aa 16" and the configured start sequences shown in the table below. Start sequences begin to be evaluated when the first 68H character is successfully received. Upon successfully receiving the fourth character (the second 68H), then start condition 1 is satisfied. Once the start conditions are satisfied, the evaluation of the end conditions begins.

The start sequence processing can be terminated due to various parity, framing, or inter-character timing errors. These errors result in no received message, because the start condition was not satisfied.

Table 12- 16 Start conditions

Start condition	First Character	First Character +1	First Character +2	First Character +3	First Character +4
1	**68**H	xx	xx	**68**H	xx
2	10H	aaH	xx	xx	xx
3	dcH	aaH	xx	xx	xx
4	e5H	xx	xx	xx	xx

End conditions for the RCV_PTP instruction

The end of a message is determined by the specification of end conditions. The end of a message is determined by the first occurrence of one or more configured end conditions. The section "Message end conditions" in the topic "Configuring receive parameters (Page 553)" describes the end conditions that you can configure in the RCV_CFG instruction.

You can configure the end conditions in either the properties of the communication interface in the device configuration, or from the RCV_CFG instruction. Whenever the CPU transitions from STOP to RUN, the receive parameters (both start and end conditions) return to the device configuration settings. If the STEP 7 user program executes RCV_CFG, then the settings are changed to the RCV_CFG conditions.

Parameter CONDITIONS data type structure part 2 (end conditions)

Table 12- 17 CONDITIONS structure for end conditions

Parameter	Parameter type	Data type	Description
ENDCOND	IN	UInt 0	This parameter specifies message end condition: • 01H - Response time • 02H - Message time • 04H - Inter-character gap • 08H - Maximum length • 10H - N + LEN + M • 20H - Sequence
MAXLEN	IN	UInt 1	Maximum message length: Only used when the maximum length end condition is selected. 1 to 1024 bytes
N	IN	UInt 0	Byte position within the message of the length field. Only used with the N + LEN + M end condition. 1 to 1022 bytes
LENGTHSIZE	IN	UInt 0	Size of the byte field (1, 2, or 4 bytes). Only used with the N + LEN + M end condition.
LENGTHM	IN	UInt 0	Specify the number of characters following the length field that are not included in the value of the length field. This is only used with the N + LEN + M end condition. 0 to 255 bytes
RCVTIME	IN	UInt 200	Specify how long to wait for the first character to be received. The receive operation will be terminated with an error if a character is not successfully received within the specified time. This is only used with the response time condition. (0 to 65535 bit times with an 8 second maximum) This parameter is not a message end condition since evaluation terminates when the first character of a response is received. It is an end condition only in the sense that it terminates a receiver operation because no response is received when a response is expected. You must select a separate end condition.
MSGTIME	IN	UInt 200	Specify how long to wait for the entire message to be completely received once the first character has been received. This parameter is only used when the message timeout condition is selected. (0 to 65535 milliseconds)
CHARGAP	IN	UInt 12	Specify the number of bit times between characters. If the number of bit times between characters exceeds the specified value, then the end condition will be satisfied. This is only used with the inter-character gap condition. (0 to 65535 bit times up to 8 second maximum)

Parameter	Parameter type	Data type	Description
ENDSEQ1CTL	IN	Byte B#16#0	Sequence 1 ignore/compare control for each character: These are the enabling bits for each character for the end sequence. Character 1 is bit 0, character 2 is bit 1, …, character 5 is bit 4. Disabling the bit associated with a character means any character will match, in this sequence position.
ENDSEQ1	IN	Char[5] 0	Sequence 1 start characters (5 characters)

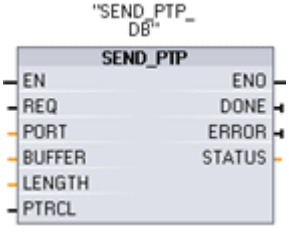
Table 12- 18 Condition codes

STATUS (W#16#....)	Description
80C0	Illegal start condition selected
80C1	Illegal end condition selected, no end condition selected
80C2	Receive interrupt enabled and this is not possible.
80C3	Maximum length end condition is enabled and max length is 0 or > 1024.
80C4	Calculated length is enabled and N is >= 1023.
80C5	Calculated length is enabled and length is not 1, 2 or 4.
80C6	Calculated length is enabled and M value is > 255.
80C7	Calculated length is enabled and calculated length is > 1024.
80C8	Response timeout is enabled and response timeout is zero.
80C9	Inter-character gap timeout is enabled and it is zero.
80CA	Idle line timeout is enabled and it is zero.
80CB	End sequence is enabled but all chars are "don't care".
80CC	Start sequence (any one of 4) is enabled but all characters are "don't care".

12.3.1.5 SEND_PTP instruction

Table 12- 19 SEND_PTP (Send Point-to-Point data) instruction

LAD / FBD	SCL	Description

	`"SEND_PTP_DB"(`
	`REQ:=_bool_in_,`
	`PORT:=_uint_in_,`
	`BUFFER:=_variant_in_,`
	`LENGTH:=_uint_in_,`
	`PTRCL:=_bool_in_,`
	`DONE=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_);`
		SEND_PTP initiates the transmission of the data and transfers the assigned buffer to the communication interface. The CPU program continues while the CM or CB sends the data at the assigned baud rate. Only one send operation can be pending at a given time. The CM or CB returns an error if a second SEND_PTP is executed while the CM or CB is already transmitting a message.

[1] STEP 7 automatically creates the DB when you insert the instruction.

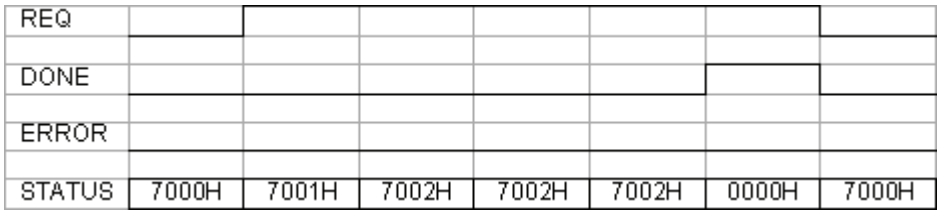
Table 12- 20 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Activates the requested transmission on the rising edge of this transmission enable input. This initiates transfer of the contents of the buffer to the Point-to-Point communication interface. (Default value: False)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table. (Default value: 0)
BUFFER	IN	Variant	This parameter points to the starting location of the transmit buffer. (Default value: 0) **Note:** Boolean data or Boolean arrays are not supported.
LENGTH	IN	UInt	Transmitted frame length in bytes (Default value: 0) When transmitting a complex structure, always use a length of 0.
PTRCL	IN	Bool	This parameter selects the buffer as normal point-to-point or specific Siemens-provided protocols that are implemented within the attached CM or CB. (Default value: False) FALSE = user program controlled point-to-point operations. (only valid option)
DONE	OUT	Bool	TRUE for one scan, after the last request was completed with no error
ERROR	OUT	Bool	TRUE for one scan, after the last request was completed with an error
STATUS	OUT	Word	Execution condition code (Default value: 0)

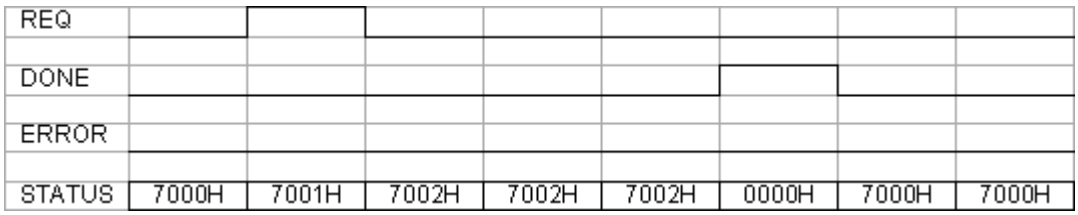
While a transmit operation is in progress, the DONE and ERROR outputs are FALSE. When a transmit operation is complete, either the DONE or the ERROR output will be set TRUE to show the status of the transmit operation. While DONE or ERROR is TRUE, the STATUS output is valid.

The instruction returns a status of 16#7001 if the communication interface accepts the transmit data. Subsequent SEND_PTP executions return 16#7002, if the CM or CB is still busy transmitting. When the transmit operation is complete, the CM or CB returns the status of the transmit operation as 16#0000 (if no errors occurred). Subsequent executions of SEND_PTP with REQ low return a status of 16#7000 (not busy).

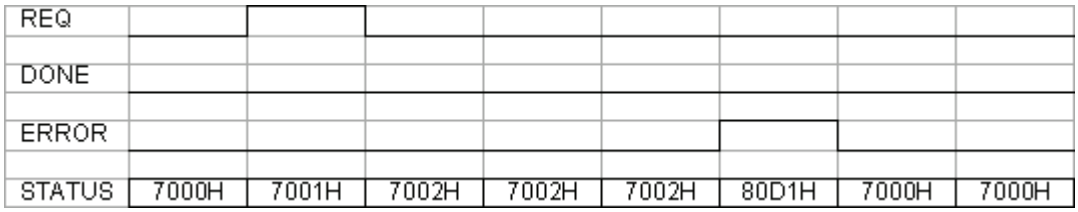
The following diagrams show the relationship of the output values to REQ. This assumes that the instruction is called periodically to check for the status of the transmission process. In the diagram below, it is assumed that the instruction is called every scan (represented by the STATUS values).



The following diagram shows how the DONE and STATUS parameters are valid for only one scan if the REQ line is pulsed (for one scan) to initiate the transmit operation.



The following diagram shows the relationship of DONE, ERROR and STATUS parameters when there is an error.



The DONE, ERROR and STATUS values are only valid until SEND_PTP executes again with the same instance DB.

Table 12- 21 Condition codes

STATUS (W#16#....)	Description
80D0	New request while transmitter active
80D1	Transmit aborted because of no CTS within wait time
80D2	Transmit aborted because of no DSR from the DCE device
80D3	Transmit aborted because of queue overflow (transmit more than 1024 bytes)
80D5	Reverse bias signal (wire break condition)
833A	The DB for the BUFFER parameter does not exist.

Interaction of the LENGTH and BUFFER parameters for SEND_PTP

The minimum size of data that can be transmitted by the SEND_PTP instruction is one byte. The BUFFER parameter determines the size of the data to be transmitted. You cannot use the data type Bool or arrays of Bool for the BUFFER parameter.

You can always set the LENGTH parameter to 0 and ensure that SEND_PTP sends the entire data structure represented by the BUFFER parameter. If you only want to send part of a data structure in the BUFFER parameter, you can set LENGTH as follows:

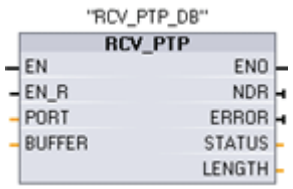
Table 12- 22 LENGTH and BUFFER parameters

LENGTH	BUFFER	Description
= 0	Not used	The complete data is sent as defined at the BUFFER parameter. You do not need to specify the number of transmitted bytes when LENGTH = 0.
> 0	Elementary data type	The LENGTH value must contain the byte count of this data type. For example, for a Word value, the LENGTH must be two. For a Dword or Real, the LENGTH must be four. Otherwise, nothing is transferred and the error 8088H is returned.
	Structure	The LENGTH value can contain a byte count less than the complete byte length of the structure, in which case only the first LENGTH bytes of the structure are sent from the BUFFER. Since the internal byte organization of a structure cannot always be determined, you might get unexpected results. In this case, use a LENGTH of 0 to send the complete structure.
	Array	The LENGTH value must contain a byte count that is less than the complete byte length of the array and which must be a multiple of the data element byte count. For example, the LENGTH parameter for an array of Words must be a multiple of two and for an array of Reals, a multiple of four. When LENGTH is specified, the number of array elements which are contained in LENGTH bytes is transferred. If your BUFFER, for example, contains an array of 15 Dwords (60 total bytes), and you specify a LENGTH of 20, then the first five Dwords in the array are transferred. The LENGTH value must be a multiple of the data element byte count. Otherwise, STATUS = 8088H, ERROR = 1, and no transmission occurs.
	String	The LENGTH parameter contains the number of characters to be transmitted. Only the characters of the String are transmitted. The maximum and actual length bytes of the String are not transmitted.

12.3.1.6 RCV_PTP instruction

Table 12- 23 RCV_PTP (Receive Point-to-Point) instruction

LAD / FBD	SCL	Description

	`"RCV_PTP_DB"(`
	`EN_R:=_bool_in_,`
	`PORT:=_uint_in_,`
	`BUFFER:=_variant_in_,`
	`NDR=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`LENGTH=>_uint_out_);`
		RCV_PTP checks for messages that have been received in the CM or CB. If a message is available, it will be transferred from the CM or CB to the CPU. An error returns the appropriate STATUS value.

[1] STEP 7 automatically creates the DB when you insert the instruction.

Table 12- 24 Data types for the parameters

Parameter and type		Data type	Description
EN_R	IN	Bool	When this input is TRUE and a message is available, the message is transferred from the CM or CB to the BUFFER. When EN_R is FALSE, the CM or CB is checked for messages and NDR, ERROR and STATUS output are updated, but the message is not transferred to the BUFFER. (Default value: 0)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table. (Default value: 0)
BUFFER	IN	Variant	This parameter points to the starting location of the receive buffer. This buffer should be large enough to receive the maximum length message. Boolean data or Boolean arrays are not supported. (Default value: 0)
NDR	OUT	Bool	TRUE for one execution when new data is ready and operation is complete with no errors.
ERROR	OUT	Bool	TRUE for one execution after the operation was completed with an error.
STATUS	OUT	Word	Execution condition code (Default value: 0)
LENGTH	OUT	UInt	Length of the returned message in bytes (Default value: 0)

The STATUS value is valid when either NDR or ERROR is TRUE. The STATUS value provides the reason for termination of the receive operation in the CM or CB. This is typically a positive value, indicating that the receive operation was successful and that the receive process terminated normally. If the STATUS value is negative (the Most Significant Bit of the hexadecimal value is set), the receive operation was terminated for an error condition such as parity, framing, or overrun errors.

Each PtP communication interface can buffer up to a maximum of 1024 bytes. This could be one large message or several smaller messages. If more than one message is available in the CM or CB, the RCV_PTP instruction returns the oldest message available. A subsequent RCV_PTP instruction execution returns the next oldest message available.

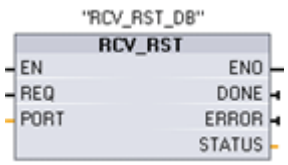
Table 12- 25 Condition codes

STATUS (W#16#...)	Description
0000	No buffer present
80E0	Message terminated because the receive buffer is full
80E1	Message terminated due to parity error
80E2	Message terminated due to framing error
80E3	Message terminated due to overrun error
80E4	Message terminated because calculated length exceeds buffer size
80E5	Reverse bias signal (wire break condition)
0094	Message terminated due to received maximum character length
0095	Message terminated because of message timeout
0096	Message terminated because of inter-character timeout
0097	Message terminated because of response timeout
0098	Message terminated because the "N+LEN+M" length condition was satisfied

STATUS (W#16#...)	Description
0099	Message terminated because of end sequence was satisfied
833A	The DB for the BUFFER parameter does not exist.

12.3.1.7 RCV_RST instruction

Table 12- 26 RCV_RST (Receiver Reset) instruction

LAD / FBD	SCL	Description
	"RCV_RST_DB"(REQ:=_bool_in_, PORT:=_uint_in_, DONE=>_bool_out_, ERROR=>_bool_out_, STATUS=>_word_out_);	RCV_RST clears the receive buffers in the CM or CB.

[1] STEP 7 automatically creates the DB when you insert the instruction.

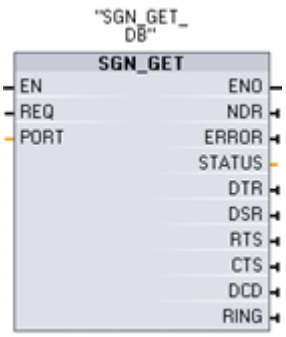
Table 12- 27 Data types for parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Activates the receiver reset on the rising edge of this enable input (Default value: False)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table. (Default value: 0)
DONE	OUT	Bool	When TRUE for one scan, indicates that the last request was completed without errors.
ERROR	OUT	Bool	When TRUE, shows that the last request was completed with errors. Also, when this output is TRUE, the STATUS output will contain related error codes.
STATUS	OUT	Word	Error code (Default value: 0)

12.3.1.8 SGN_GET instruction

Table 12- 28 SGN_GET (Get RS232 signals) instruction

LAD / FBD	SCL	Description

	`"SGN_GET_DB"(`
	`REQ:=_bool_in_,`
	`PORT:=_uint_in_,`
	`NDR=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`DTR=>_bool_out_,`
	`DSR=>_bool_out_,`
	`RTS=>_bool_out_,`
	`CTS=>_bool_out_,`
	`DCD=>_bool_out_,`
	`RING=>_bool_out_);`
		SGN_GET reads the current states of RS232 communication signals.
		This function is valid only for the RS232 CM.

[1] STEP 7 automatically creates the DB when you insert the instruction.

Table 12- 29 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Get RS232 signal state values on the rising edge of this input (Default value: False)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table.
NDR	OUT	Bool	TRUE for one scan, when new data is ready and the operation is complete with no errors
ERROR	OUT	Bool	TRUE for one scan, after the operation was completed with an error
STATUS	OUT	Word	Execution condition code (Default value: 0)
DTR	OUT	Bool	Data terminal ready, module ready (output). Default value: False
DSR	OUT	Bool	Data set ready, communication partner ready (input). Default value: False
RTS	OUT	Bool	Request to send, module ready to send (output). Default value: False
CTS	OUT	Bool	Clear to send, communication partner can receive data (input). Default value: False
DCD	OUT	Bool	Data carrier detect, receive signal level (always False, not supported)
RING	OUT	Bool	Ring indicator, indication of incoming call (always False, not supported)

Table 12- 30 Condition codes

STATUS (W#16#....)	Description
80F0	CM or CB is RS485 and no signals are available

12.3.1.9 SGN_SET instruction

Table 12- 31 SGN_SET (Set RS232 signals) instruction

LAD / FBD	SCL	Description

	`"SGN_SET_DB"(`
	`REQ:=_bool_in_,`
	`PORT:=_uint_in_,`
	`SIGNAL:=_byte_in_,`
	`RTS:=_bool_in_,`
	`DTR:=_bool_in_,`
	`DSR:=_bool_in_,`
	`DONE=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_);`
		SGN_SET sets the states of RS232 communication signals.
		This function is valid only for the RS232 CM.

[1] STEP 7 automatically creates the DB when you insert the instruction.

Table 12- 32 Data types for parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Start the set RS232 signals operation, on the rising edge of this input (Default value: False)
PORT	IN	PORT	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table. (Default value: 0)
SIGNAL	IN	Byte	Selects which signal to set: (multiple allowed). Default value: 0 • 01H = Set RTS • 02H = Set DTR • 04H = Set DSR
RTS	IN	Bool	Request to send, module ready to send value to set (true or false), Default value: False
DTR	IN	Bool	Data terminal ready, module ready to send value to set (true or false). Default value: False
DSR	IN	Bool	Data set ready (only applies to DCE type interfaces), not used.
DONE	OUT	Bool	TRUE for one execution after the last request was completed with no error
ERROR	OUT	Bool	TRUE for one execution after the last request was completed with an error
STATUS	OUT	Word	Execution condition code (Default value: 0)

Table 12- 33 Condition codes

STATUS (W#16#....)	Description
80F0	CM or CB is RS485 and no signals can be set
80F1	Signals cannot be set because of Hardware flow control
80F2	Cannot set DSR because module is DTE
80F3	Cannot set DTR because module is DCE

12.3.2 Configuring the communication ports

The communication interfaces can be configured by two methods:

- Use the device configuration in STEP 7 to configure the port parameters (baud and parity), the send parameters and the receive parameters. The device configuration settings are stored in the CPU. These settings are applied after a power cycle and a RUN to STOP transition.
- Use the PORT_CFG (Page 534), SEND_CFG (Page 535) and RCV_CFG (Page 537) instructions to set the parameters. The port settings set by the instructions are valid while the CPU is in RUN mode. The port settings revert to the device configuration settings after a STOP transition or power cycle.

After configuring the hardware devices (Page 109), you configure parameters for the communication interfaces by selecting one of the CMs in your rack or the CB, if configured.

The "Properties" tab of the inspector window displays the parameters of the selected CM or CB. Select "Port configuration" to edit the following parameters:

- Baud rate
- Parity
- Number of stop bits
- Flow control (RS232 only)
- Wait time

Except for flow control, which only the CM 1241 RS232 supports, the port configuration parameters are the same regardless of whether you are configuring an RS232 or an RS485 communication module or the RS485 communication board. The parameter values can differ.

The STEP 7 user program can also configure the port or change the existing configuration with the PORT_CFG instruction (Page 534).

Note

Parameter values set from the PORT_CFG instruction in the user program override port configuration settings set from the device configuration. Note that the S7-1200 does not retain parameters set from the PORT_CFG instruction in the event of power down.

Parameter	Definition
Baud rate	The default value for the baud rate is 9.6 Kbits per second. Valid choices are: 300 baud, 600 baud, 1.2 Kbits, 2.4 Kbits, 4.8 Kbits, 9.6 Kbits, 19.2 Kbits, 38.4 Kbits, 57.6 Kbits, 76.8 Kbits, and 115.2 Kbits.
Parity	The default value for parity is no parity. Valid choices are: No parity, even, odd, mark (parity bit always set to 1), and space (parity bit always set to 0).
Number of stop bits	The number of stop bits can be either one or two. The default is one.
Flow control	For the RS232 communication module, you can select either hardware or software flow control, as described in the section "Managing flow control (Page 550)". If you select hardware flow control, you can select whether the RTS signal is always on, or RTS is switched. If you select software flow control, you can define the XON and XOFF characters. The RS485 communication interfaces do not support flow control.
Wait time	Wait time specifies the time that the CM or CB waits to receive CTS after asserting RTS, or for receiving an XON after receiving an XOFF, depending on the type of flow control. If the wait time expires before the communication interface receives an expected CTS or XON, the CM or CB aborts the transmit operation and returns an error to the user program. You specify the wait time in milliseconds. The range is 0 to 65535 milliseconds.

12.3.2.1 Managing flow control

Flow control refers to a mechanism for balancing the sending and receiving of data transmissions so that no data is lost. Flow control ensures that a transmitting device is not sending more information than a receiving device can handle. Flow control can be accomplished through either hardware or software. The RS232 CM supports both hardware and software flow control. The RS485 CM and CB do not support flow control. You specify the type of flow control either when you configure the port (Page 549) or with the PORT_CFG instruction (Page 534).

Hardware flow control works through the Request-to-send (RTS) and Clear-to-send (CTS) communication signals. With the RS232 CM, the RTS signal is output from pin 7 and the CTS signal is received through pin 8. The RS232 CM is a DTE (Data Terminal Equipment) device which asserts RTS as an output and monitors CTS as an input.

Hardware flow control: RTS switched

If you enable RTS switched hardware flow control for an RS232 CM, the module sets the RTS signal active to send data. It monitors the CTS signal to determine whether the receiving device can accept data. When the CTS signal is active, the module can transmit data as long as the CTS signal remains active. If the CTS signal goes inactive, then the transmission must stop.

Transmission resumes when the CTS signal becomes active. If the CTS signal does not become active within the configured wait time, the module aborts the transmission and returns an error to the user program. You specify the wait time in the port configuration (Page 549).

The RTS switched flow control is useful for devices that require a signal that the transmit is active. An example would be a radio modem that uses RTS as a "Key" signal to energize the radio transmitter. The RTS switched flow control will not function with standard telephone modems. Use the RTS always on selection for telephone modems.

Hardware flow control: RTS always on

In RTS always on mode, the CM 1241 sets RTS active by default. A device such as a telephone modem monitors the RTS signal from the CM and utilizes this signal as a clear-to-send. The modem only transmits to the CM when RTS is active, that is, when the telephone modem sees an active CTS. If RTS is inactive, the telephone module does not transmit to the CM.

To allow the modem to send data to the CM at any time, configure "RTS always on" hardware flow control. The CM thus sets the RTS signal active all the time. The CM will not set RTS inactive even if the module cannot accept characters. The transmitting device must ensure that it does not overrun the receive buffer of the CM.

Data Terminal Block Ready (DTR) and Data Set Ready (DSR) signal utilization

The CM sets DTR active for either type of hardware flow control. The module transmits only when the DSR signal becomes active. The state of DSR is only evaluated at the start of the send operation. If DSR becomes inactive after transmission has started, the transmission will not be paused.

Software flow control

Software flow control uses special characters in the messages to provide flow control. You configure Hex characters that represent XON and XOFF.

XOFF indicates that a transmission must stop. XON indicates that a transmission can resume. XOFF and XON must not be the same character.

When the transmitting device receives an XOFF character from the receiving device, it stops transmitting. Transmitting resumes when the transmitting device receives an XON character. If it does not receive an XON character within the wait time that is specified in the port configuration (Page 549), the CM aborts the transmission and returns an error to the user program.

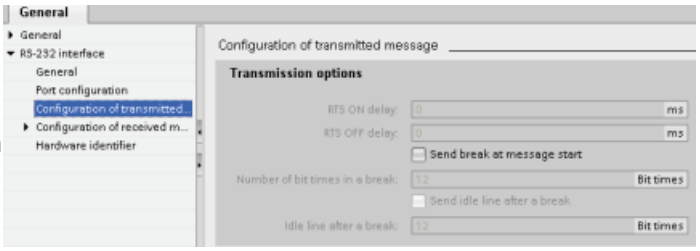
Software flow control requires full-duplex communication, as the receiving partner must be able to send XOFF to the transmitting partner while a transmission is in progress. Software flow control is only possible with messages that contain only ASCII characters. Binary protocols cannot utilize sofware flow control.

12.3.3 Configuring the transmit (send) and receive parameters

Before the CPU can engage in PtP communications, you must configure parameters for transmitting (or sending) messages and receiving messages. These parameters dictate how communications operate when messages are being transmitted to or received from a target device.

12.3.3.1 Configuring transmit (send) parameters

From the device configuration, you configure how a communication interface transmits data by specifying the "Configuration of transmitted message" properties for the selected interface.



You can also dynamically configure or change the transmit message parameters from the user program by using the SEND_CFG (Page 535) instruction.

Note

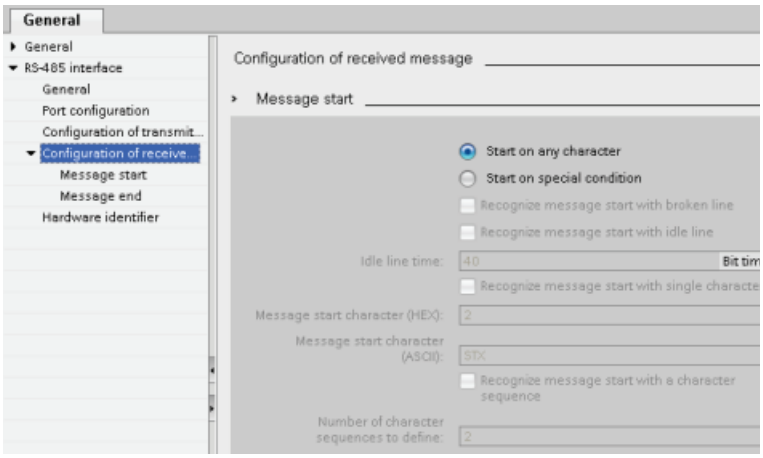
Parameter values set from the SEND_CFG instruction in the user program override the port configuration settings. Note that the CPU does not retain parameters set from the SEND_CFG instruction in the event of power down.

Parameter	Definition
RTS On delay	Specifies the amount of time to wait after activating RTS before transmission is initiated. The range is 0 to 65535 ms, with a default value of 0. This parameter is valid only when the port configuration (Page 549) specifies hardware flow control. CTS is evaluated after the RTS On delay time has expired. This parameter is applicable for RS232 modules only.
RTS Off delay	Specifies the amount of time to wait before de-activating RTS after completion of transmission. The range is 0 to 65535 ms, with a default value of 0. This parameter is valid only when the port configuration (Page 549) specifies hardware flow control. This parameter is applicable for RS232 modules only.

Parameter	Definition
Send break at message start Number of bit times in a break	Specifies that upon the start of each message, a break will be sent after the RTS On delay (if configured) has expired and CTS is active. You specify how many bit times constitute a break where the line is held in a spacing condition. The default is 12 and the maximum is 65535, up to a limit of eight seconds.
Send idle line after a break Idle line after a break	Specifies that an idle line will be sent before message start. It is sent after the break, if a break is configured. The "Idle line after a break" parameter specifies how many bit times constitute an idle line where the line is held in a marking condition. The default is 12 and the maximum is 65535, up to a limit of eight seconds.

12.3.3.2 Configuring receive parameters

From the device configuration, you configure how a communication interface receives data, and how it recognizes both the start of and the end of a message. Specify these parameters in the Receive message configuration for the selected interface.



You can also dynamically configure or change the receive message parameters from the user program by using the RCV_CFG instruction (Page 537).

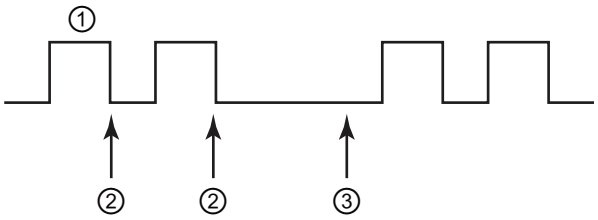
Note

Parameter values set from the RCV_CFG instruction in the user program override the port configuration settings. Note that the CPU does not retain parameters set from the RCV_CFG instruction in the event of power down.

Message start conditions

You can determine how the communication interface recognizes the start of a message. The start characters and the characters comprising the message go into the receive buffer until a configured end condition is met.

You can specify multiple start conditions. If you specify more than one start condition, all of the start conditions must be met before the message is considered started. For example, if you configure an idle line time and a specific start character, the CM or CB will first look for the idle line time requirement to be met and then the CM will look for the specified start character. If some other character is received (not the specified start character), the CM or CB will restart the start of message search by again looking for an idle line time.

Parameter	Definition
Start on Any Character	The Any Character condition specifies that any successfully received character indicates the start of a message. This character is the first character within a message.
Line Break	The Line Break conditions specifies that a message receive operation starts after a break character is received.
Idle Line	The Idle Line condition specifies that a message reception starts once the receive line has been idle or quiet for the number of specified bit times. Once this condition occurs, the start of a message begins.  ① Characters ② Restarts the idle line timer ③ Idle line is detected and message receive is started
Special condition: Recognize message start with single character	Specifies that a particular character indicates the start of a message. This character is then the first character within a message. Any character that is received before this specific character is discarded. The default character is STX.
Special condition: Recognize message start with a character sequence	Specifies that a particular character sequence from up to four configured sequences indicates the start of a message. For each sequence, you can specify up to five characters. For each character position, you specify either a specific hex character, or that the character is ignored in sequence matching (wild-card character). The last specific character of a character sequence terminates that start condition sequence. Incoming sequences are evaluated against the configured start conditions until a start condition has been satisfied. Once the start sequence has been satisfied, evaluation of end conditions begins. You can configure up to four specific character sequences. You use a multiple-sequence start condition when different sequences of characters can indicate the start of a message. If any one of the character sequences is matched, the message is started.

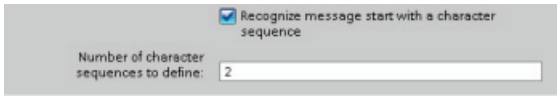
The order of checking start conditions is:

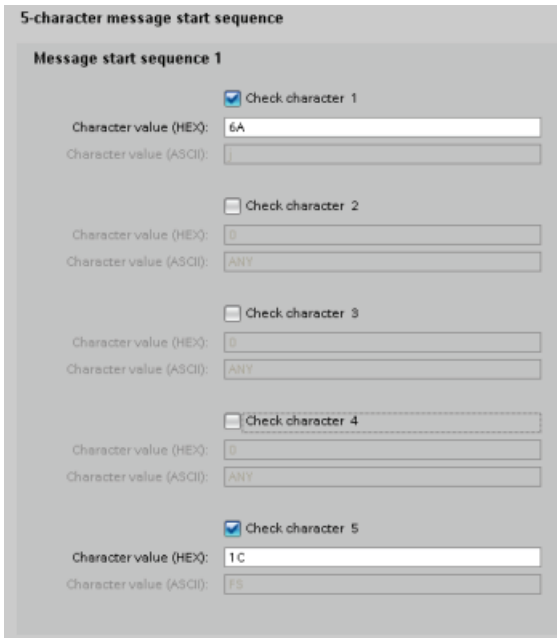
- Idle line
- Line break
- Characters or character sequences

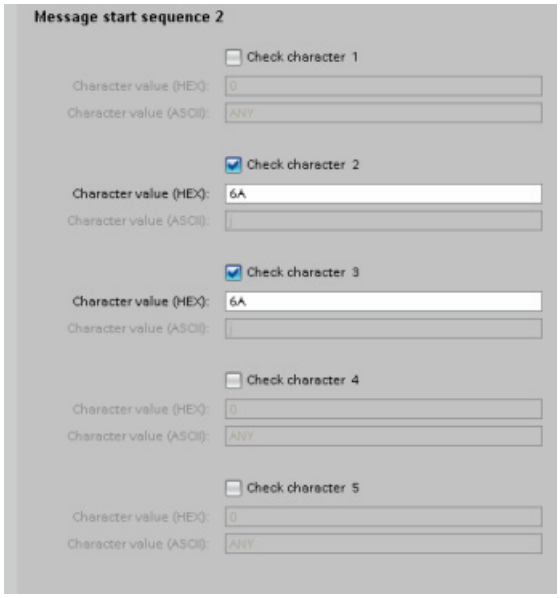
While checking for multiple start conditions, if one of the conditions is not met, the CM or CB will restart the checking with the first required condition. After the CM or CB establishes that the start conditions have been met, it begins evaluating end conditions.

Sample configuration - start message on one of two character sequences

Consider the following start message condition configuration:







With this configuration, the start condition is satisfied when either pattern occurs:

- When a five-character sequence is received where the first character is 0x6A and the fifth character is 0x1C. The characters at positions 2, 3, and 4 can be any character with this configuration. After the fifth character is received, evaluation of end conditions begins.
- When two consecutive 0x6A characters are received, preceded by any character. In this case, evaluation of end conditions begins after the second 0x6A is received (3 characters). The character preceding the first 0x6A is included in the start condition.

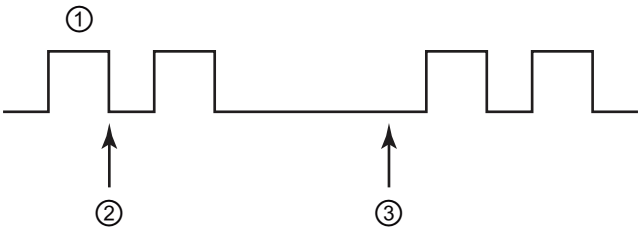
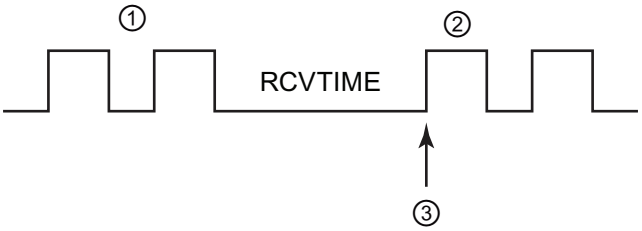
Example sequences that would satisfy this start condition are:

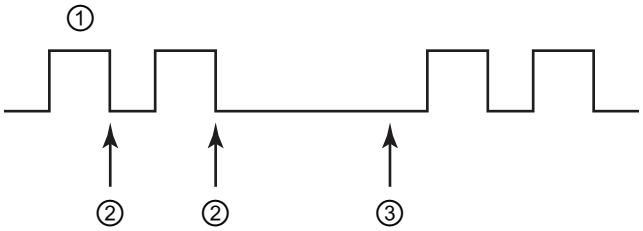
- <any character> 6A 6A
- 6A 12 14 18 1C
- 6A 44 A5 D2 1C

Message end conditions

You also configure how the communication interface recognizes the end of a message. You can configure multiple message end conditions. If any one of the configured conditions occurs, the message ends.

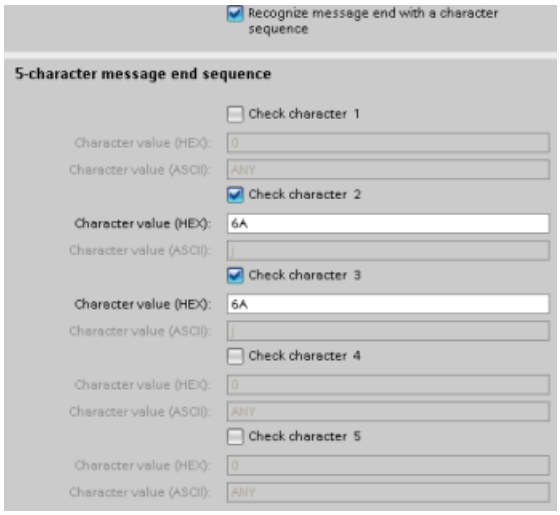
For example, you could specify an end condition with an end of message timeout of 300 milliseconds, an inter-character timeout of 40 bit times, and a maximum length of 50 bytes. The message will end if the message takes longer than 300 milliseconds to receive, or if the gap between any two characters exceeds 40 bit times, or if 50 bytes are received.

Parameter	Definition
Recognize message end by message timeout	The message end occurs when the configured amount of time to wait for the message end has expired. The message timeout period begins when a start condition has been satisfied. The default is 200 ms and the range is 0 to 65535 ms.  ① Received characters ②Start Message condition satisfied: message timer starts ③ Message timer expires and terminates the message
Recognize message end by response timeout	The message end occurs when the configured amount of time to wait for a response expires before a valid start sequence is received. The response timeout period begins when a transmission ends and the CM or CB begins the receive operation. The default response timeout is 200 ms and the range is 0 to 65535 ms. If a character is not received within the response time period, RCVTIME, then an error is returned to the corresponding RCV_PTP instruction. The response timeout does not define a specific end condition. It only specifies that a character must be successfully received within the specified time. You must configure another end condition to indicate the actual end of a message.  ① Transmitted characters ② Received characters ③ First character must be successfully received by this time.

Parameter	Definition
Recognize message end by inter-character gap	The message end occurs when the maximum configured timeout between any two consecutive characters of a message has expired. The default value for the inter-character gap is 12 bit times and the maximum number is 65535 bit times, up to a maximum of eight seconds.  ① Received characters ② Restarts the intercharacter timer ③ The intercharacter timer expires and terminates the message.
Recognize message end by max length	The message end occurs when the configured maximum number of characters has been received. The valid range for maximum length is 1 to 1023. This condition can be used to prevent a message buffer overrun error. When this end condition is combined with timeout end conditions and the timeout condition occurs, any valid received characters are provided even if the maximum length is not reached. This allows support for varying length protocols when only the maximum length is known.
Read message length from message	The message itself specifies the length of the message. The message end occurs when a message of the specified length has been received. The method for specifying and interpreting the message length is described below.
Recognize message end with a character	The message end occurs when a specified character is received.
Recognize message end with a character sequence	The message end occurs when a specified character sequence is received. You can specify a sequence of up to five characters. For each character position, you specify either a specific hex character, or that the character is ignored in sequence matching. Leading characters that are ignored characters are not part of the end condition. Trailing characters that are ignored characters are part of the end condition.

Sample configuration - end message with a character sequence

Consider the following end message condition configuration:



In this case, the end condition is satisfied when two consecutive 0x7A characters are received, followed by any two characters. The character preceding the 0x7A 0x7A pattern is not part of the end character sequence. Two characters following the 0x7A 0x7A pattern are required to terminate the end character sequence. The values received at character positions 4 and 5 are irrelevant, but they must be received to satisfy the end condition.

Specification of message length within the message

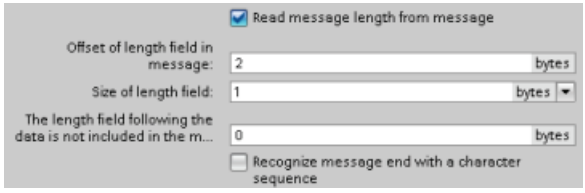
When you select the special condition where the message length is included in the message, you must provide three parameters that define information about the message length.

The actual message structure varies according to the protocol in use. The three parameters are as follows:

- n: the character position (1-based) within the message that starts the length specifier
- Length size: The number of bytes (one, two, or four) of the length specifier
- Length m: the number of characters following the length specifier that are not included in the length count

The ending characters do not need to be contiguous. The "Length m" value can be used to specify the length of a checksum field whose size is not included in the length field.

These fields appear in the Receive message configuration of the device properties:



Example 1: Consider a message structured according to the following protocol:

STX	Len (n)	Characters 3 to 14 counted by the length												
		ADR	PKE		INDEX		PWD		STW		HSW		BCC
1	2	3	4	5	6	7	8	9	10	11	12	13	14
STX	0x0C	xx	xxxx		xxxx		xxxx		xxxx		xxxx		xx

Configure the receive message length parameters for this message as follows:

- n = 2 (The message length starts with byte 2.)
- Length size = 1 (The message length is defined in one byte.)
- Length m = 0 (There are no additional characters following the length specifier that are not counted in the length count. Twelve characters follow the length specifier.)

In this example, the characters from 3 to 14 inclusive are the characters counted by Len (n).

Example 2: Consider another message structured according to the following protocol:

SD1	Len (n)	Len (n)	SD2	Characters 5 to 10 counted by length						FCS	ED
				DA	SA	FA	Data unit=3 bytes				
1	2	3	4	5	6	7	8	9	10	11	12
xx	0x06	0x06	xx	xx	xx	xx	xx	xx	xx	xx	xx

Configure the receive message length parameters for this message as follows:

- n = 3 (The message length starts at byte 3.)
- Length size = 1 (The message length is defined in one byte.)
- Length m = 3 (There are three characters following the length specifier that are not counted in the length. In the protocol of this example, the characters SD2, FCS, and ED are not counted in the length count. The other six characters are counted in the length count; therefore the total number of characters following the length specifier is nine.)

In this example, the characters from 5 to 10 inclusive are the characters counted by Len (n).

12.3.4 Programming the PtP communications

STEP 7 provides extended instructions that enable the user program to perform Point-to-Point communications with a protocol designed and specified in the user program. These instructions can be considered in two categories:

- Configuration instructions
- Communication instructions

Configuration instructions

Before your user program can engage in PtP communication, you must configure the communication interface port and the parameters for sending data and receiving data.

You can perform the port configuration and message configuration for each CM or CB through the device configuration or through these instructions in your user program:

- PORT_CFG (Page 534)
- SEND_CFG (Page 535)
- RCV_CFG (Page 537)

Communication instructions

The PtP communication instructions enable the user program to send messages to and receive messages from the communication interfaces. For information about transferring data with these instructions, see the section on data consistency (Page 143).

All of the PtP functions operate asynchronously. The user program can use a polling architecture to determine the status of transmissions and receptions. SEND_PTP and RCV_PTP can execute concurrently. The communication modules and communication board buffer the transmit and receive messages as necessary up to a maximum buffer size of 1024 bytes.

The CMs and CB send messages to and receive messages from the actual point-to-point devices. The message protocol is in a buffer that is either received from or sent to a specific communication port. The buffer and port are parameters of the send and receive instructions:

- SEND_PTP (Page 541)
- RCV_PTP (Page 544)

Additional instructions provide the capability to reset the receive buffer, and to get and set specific RS232 signals:

- RCV_RST (Page 546)
- SGN_GET (Page 547)
- SGN_SET (Page 548)

12.3.4.1 Polling architecture

The S7-1200 point-to-point instructions must be called cyclically/periodically to check for received messages. Polling the send will tell the user program when the transmit has completed.

Polling architecture: master

The typical sequence for a master is as follows:

1. A SEND_PTP instruction initiates a transmission to the CM or CB.
2. The SEND_PTP instruction is executed on subsequent scans to poll for the transmit complete status.

3. When the SEND_PTP instruction indicates that the transmission is complete, the user code can prepare to receive the response.
4. The RCV_PTP instruction is executed repeatedly to check for a response. When the CM or CB has collected a response message, the RCV_PTP instruction copies the response to the CPU and indicates that new data has been received.
5. The user program can process the response.
6. Go to step 1 and repeat the cycle.

Polling architecture: slave

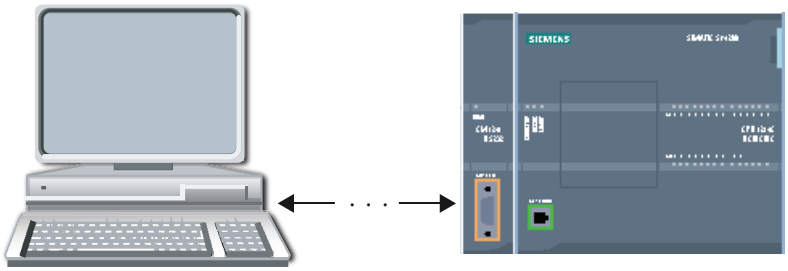
The typical sequence for a slave is as follows:

1. The user program executes the RCV_PTP instruction every scan.
2. When the CM or CB has received a request, the RCV_PTP instruction indicates that new data is ready and the request is copied into the CPU.
3. The user program services the request and generates a response.
4. Use a SEND_PTP instruction to send the response back to the master.
5. Repeatedly execute SEND_PTP to be sure the transmit occurs.
6. Go to step 1 and repeat the cycle.

The slave must be responsible for calling RCV_PTP frequently enough to receive a transmission from the master before the master times out while waiting for a response. To accomplish this task, the user program can call RCV_PTP from a cyclic OB, where the cycle time is sufficient to receive a transmission from the master before the timeout period expires. If you set the cycle time for the OB to provide for two executions within the timeout period of the master, the user program can receive transmissions without missing any.

12.3.5 Example: Point-to-Point communication

In this example, an S7-1200 CPU communicates to a PC with a terminal emulator through a CM 1241 RS232 module. The point-to-point configuration and STEP 7 program in this example illustrate how the CPU can receive a message from the PC and echo the message back to the PC.



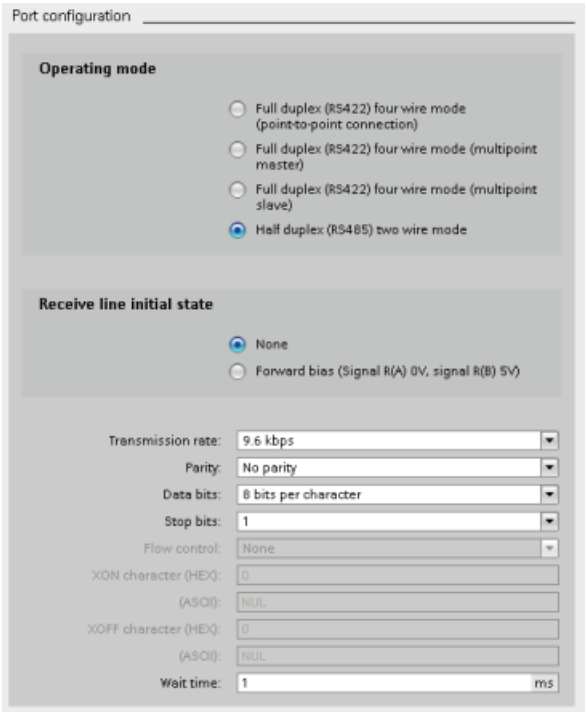
You must connect the communication interface of the CM 1241 RS232 module to the RS232 interface of the PC, which is normally COM1. Because both of these ports are Data Terminal Equipment (DTE), you must switch the receive and transmit pins (2 and 3) when connecting the two ports, which you can accomplish by either of the following methods:

- Use a NULL modem adapter to swap pins 2 and 3 together with a standard RS232 cable.
- Use a NULL modem cable, which already has pins 2 and 3 swapped. You can usually identify a NULL modem cable as one with two female 9-pin D connector ends.

12.3.5.1 Configuring the communication module

You can configure the CM 1241 from the Device configuration in STEP 7 or with user program instructions. This example uses the Device configuration method.

- Port configuration: Click the communication port of the CM module from the Device configuration, and configure the port as shown:

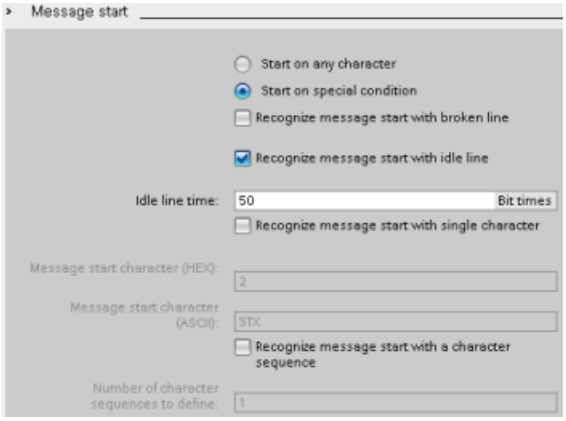


Note

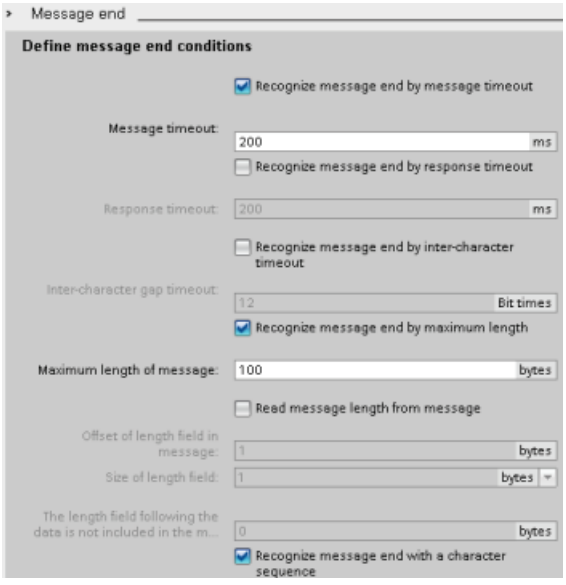
The configuration settings for "Operating mode" and "Receive line initial state" are only applicable for the CM 1241 (RS422/RS485) module. The other CM 1241 modules do not have these port configuration settings.

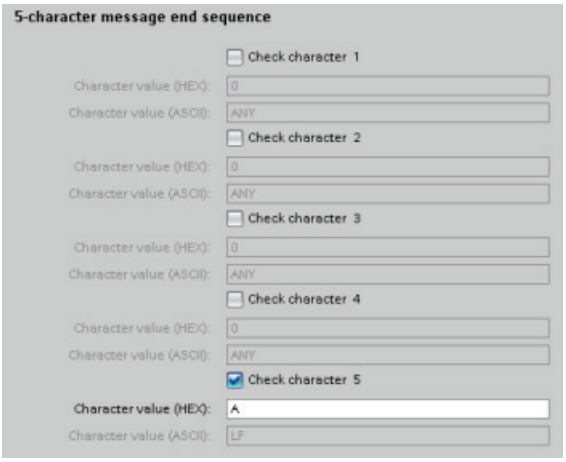
- Transmit message configuration: Accept the default for transmit message configuration. No break is to be sent at message start.

- Receive message start configuration: Configure the CM 1241 to start receiving a message when the communication line is inactive for at least 50 bit times (about 5 milliseconds at 9600 baud = 50 * 1/9600):



- Receive message end configuration: Configure the CM 1241 to end a message when it receives a maximum of 100 bytes or a linefeed character (10 decimal or A hexadecimal). The end sequence allows up to five end characters in sequence. The fifth character in the sequence is the linefeed character. The preceding four end sequence characters are "don't care" or unselected characters. The CM 1241 does not evaluate the "don't care" characters but looks for a linefeed character preceded by zero or more "don't care" characters to indicate the message end.



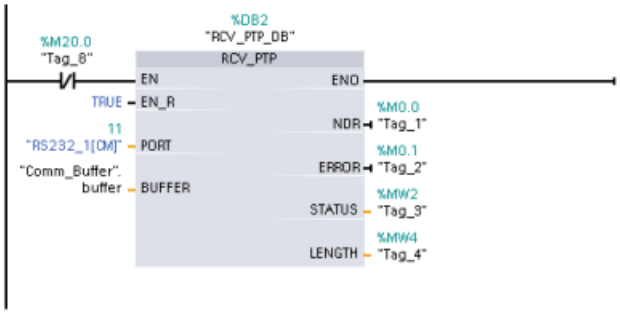


12.3.5.2 Programming the STEP 7 program

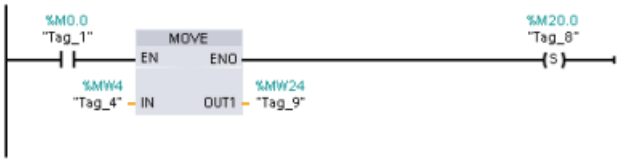
The example program uses a global data block for the communication buffer, a RCV_PTP instruction (Page 544) to receive data from the terminal emulator, and a SEND_PTP instruction (Page 541) to echo the buffer back to the terminal emulator. To program the example, add the data block configuration and program OB1 as described below.

Global data block "Comm_Buffer": Create a global data block (DB) and name it "Comm_Buffer". Create one value in the data block called "buffer" with a data type of "array [0 .. 99] of byte".

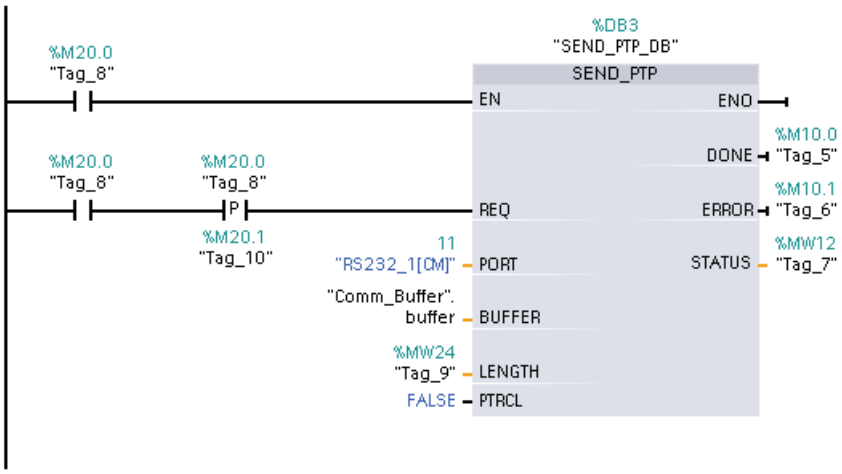
Network 1: Enable the RCV_PTP instruction whenever SEND_PTP is not active. Tag_8 at MW20.0 indicates when sending is complete in Network 4, and when the communication module is thus ready to receive a message.



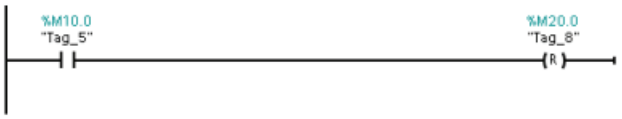
Network 2: Use the NDR value (Tag_1 at M0.0) set by the RCV_PTP instruction to make a copy of the number of bytes received and to set a flag (Tag_8 at M20.0) to trigger the SEND_PTP instruction.



Network 3: Enable the SEND_PTP instruction when the M20.0 flag is set. Also use this flag to set the REQ input to TRUE for one scan. The REQ input tells the SEND_PTP instruction that a new request is to be transmitted. The REQ input must only be set to TRUE for one execution of SEND_PTP. The SEND_PTP instruction is executed every scan until the transmit completes. The transmit is complete when the last byte of the message has been transmitted from the CM 1241. When the transmit is complete, the DONE output (Tag_5 at M10.0) is set TRUE for one execution of SEND_PTP.



Network 4: monitor the DONE output of SEND_PTP and reset the transmit flag (Tag_8 at M20.0) when the transmit operation is complete. When the transmit flag is reset, the RCV_PTP instruction in Network 1 is enabled to receive the next message.



12.3.5.3 Configuring the terminal emulator

You must set up the terminal emulator to support the example program. You can use most any terminal emulator on your PC, such as HyperTerminal. Make sure that the terminal emulator is in the disconnected mode before editing the settings as follows:

1. Set the terminal emulator to use the RS232 port on the PC (normally COM1).
2. Configure the port for 9600 baud, 8 data bits, no parity (none), 1 stop bit and no flow control.
3. Change the settings of the terminal emulator to emulate an ANSI terminal.
4. Configure the terminal emulator ASCII setup to send a line feed after every line (after the user presses the Enter key).
5. Echo the characters locally so that the terminal emulator displays what is typed.

12.3.5.4 Running the example program

To exercise the example program, follow these steps:

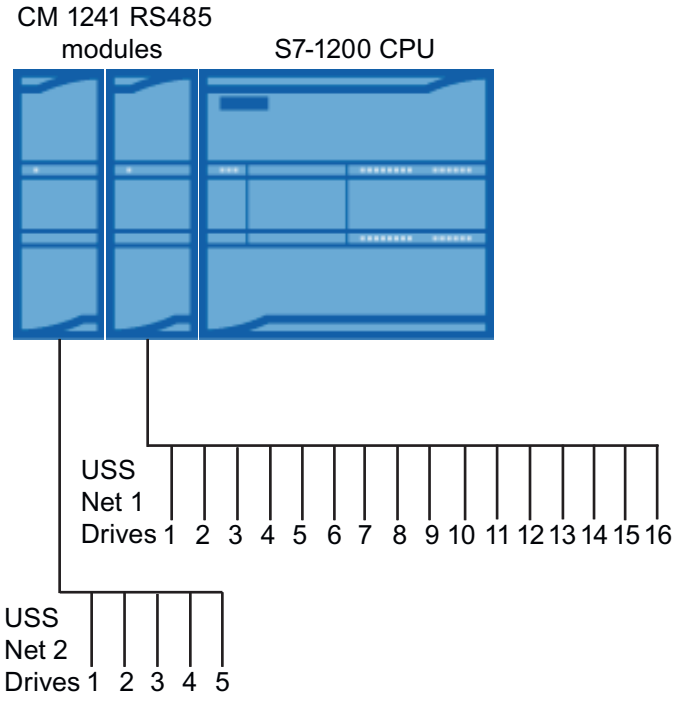
1. Download the STEP 7 program to the CPU and ensure that it is in RUN mode.
2. Click the "connect" button on the terminal emulator to apply the configuration changes and open a terminal session to the CM 1241.
3. Type characters at the PC and press Enter.

The terminal emulator sends the characters to the CM 1241 and to the CPU. The CPU program then echoes the characters back to the terminal emulator.

12.4 Universal serial interface (USS) communication

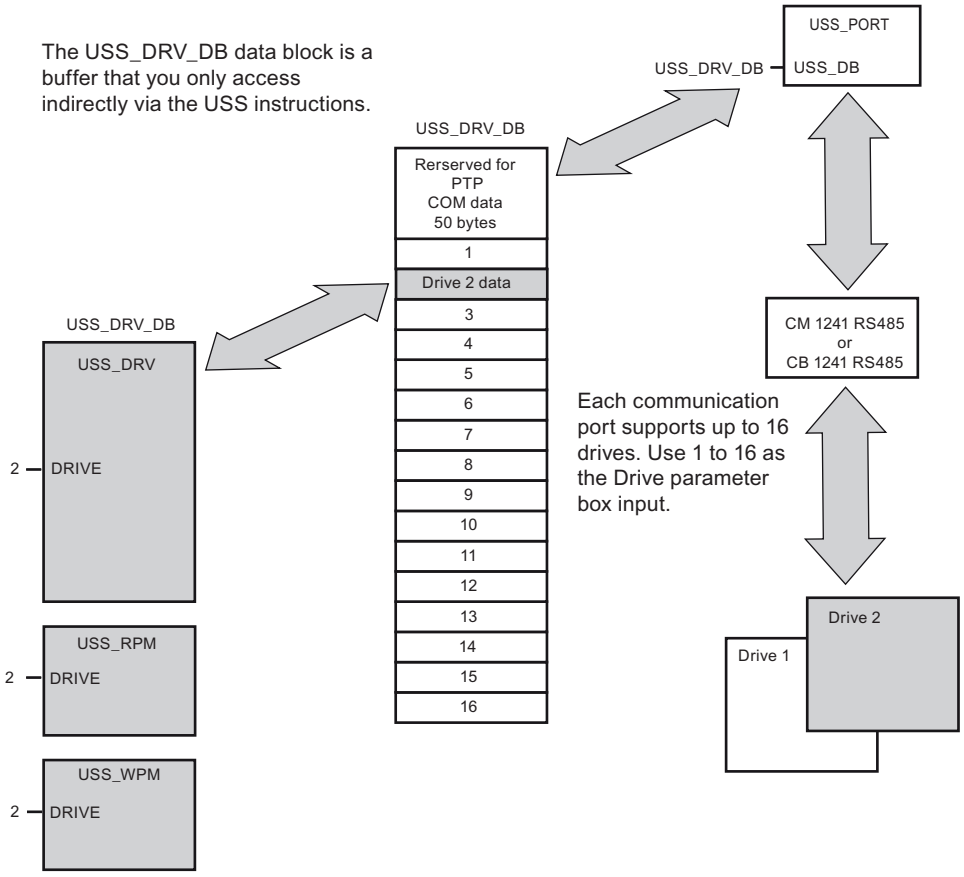
The USS instructions control the operation of motor drives which support the universal serial interface (USS) protocol. You can use the USS instructions to communicate with multiple drives through RS485 connections to CM 1241 RS485 communication modules or a CB 1241 RS485 communication board. Up to three CM 1241 RS485 modules and one CB 1241 RS485 board can be installed in a S7-1200 CPU. Each RS485 port can operate up to sixteen drives.

The USS protocol uses a master-slave network for communications over a serial bus. The master uses an address parameter to send a message to a selected slave. A slave itself can never transmit without first receiving a request to do so. Direct message transfer between the individual slaves is not possible. USS communication operates in half-duplex mode. The following USS illustration shows a network diagram for an example drive application.



12.4.1 Requirements for using the USS protocol

The four USS instructions use 1 FB and 3 FCs to support the USS protocol. One USS_PORT instance data block (DB) is used for each USS network. The USS_PORT instance data block contains temporary storage and buffers for all drives on that USS network. The USS instructions share the information in this data block.



All drives (up to 16) connected to a single RS485 port are part of the same USS network. All drives connected to a different RS485 port are part of a different USS network. Each USS network is managed using a unique data block. All instructions associated with a single USS network must share this data block. This includes all USS_DRV, USS_PORT, USS_RPM, and USS_WPM instructions used to control all drives on a single USS network.

The USS_DRV instruction is a Function Block (FB). When you place the USS_DRV instruction into the program editor, you will be prompted by the "Call options" dialog to assign a DB for this FB. If this is the first USS_DRV instruction in this program for this USS network, then you can accept the default DB assignment (or change the name if you wish) and the new DB is created for you. If however this is not the first USS_DRV instruction for this channel, then you must use the drop-down list in the "Call options" dialog to select the DB name that was previously assigned for this USS network.

Instructions USS_PORT, USS_RPM, and USS_WPM are all Functions (FCs). No DB is assigned when you place these FCs in the editor. Instead, you must assign the appropriate DB reference to the "USS_DB" input of these instructions. Double-click on the parameter field and then click on the parameter helper icon to see the available DB names).

The USS_PORT function handles the actual communication between the CPU and the drives via the Point-to-Point (PtP) RS485 communication port. Each call to this function handles one communication with one drive. Your program must call this function fast enough to prevent a communication timeout by the drives. You may call this function in a main program cycle OB or any interrupt OB.

The USS_DRV function block provides your program access to a specified drive on the USS network. Its inputs and outputs are the status and controls for the drive. If there are 16 drives on the network, your program must have at least 16 USS_DRV calls, one for each drive. These blocks should be called at the rate that is required to control the operation of the drive. Typically, you should call the USS_PORT function from a cyclic interrupt OB. The cycle time of the cyclic interrupt OB should be set to about half of the minimum call interval (As an example, 1200 baud communication should use a cyclic time of 350 ms or less).

You may only call the USS_DRV function block from a main program cycle OB.

⚠ CAUTION

Only call USS_DRV, USS_RPM, and USS_WPM from a main program cycle OB. The USS_PORT function can be called from any OB, usually from a cyclic interrupt OB.

Do not use instructions USS_DRV, USS_RPM, or USS_WPM in a higher priority OB than the corresponding USS_PORT instruction. For example, do not place the USS_PORT in the main and a USS_RPM in a cyclic interrupt OB. Failure to prevent interruption of USS_PORT execution may produce unexpected errors.

The USS_RPM and USS_WPM functions read and write the remote drive operating parameters. These parameters control the internal operation of the drive. See the drive manual for the definition of these parameters. Your program can contain as many of these functions as necessary, but only one read or write request can be active per drive, at any given time. You may only call the USS_RPM and USS_WPM functions from a main program cycle OB.

Calculating the time required for communicating with the drive

Communications with the drive are asynchronous to the S7-1200 scan cycle. The S7-1200 typically completes several scans before one drive communications transaction is completed.

The USS_PORT interval is the time required for one drive transaction. The table below shows the minimum USS_PORT interval for each communication baud rate. Calling the USS_PORT function more frequently than the USS_PORT interval will not increase the number of transactions. The drive timeout interval is the amount of time that might be taken for a transaction, if communications errors caused 3 tries to complete the transaction. By default, the USS protocol library automatically does up to 2 retries on each transaction.

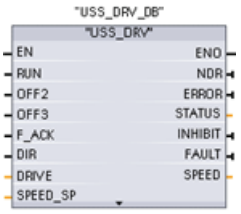
Table 12- 34 Calculating the time requirements

Baud rate	Calculated minimum USS_PORT call Interval (milliseconds)	Drive message interval timeout per drive (milliseconds)
1200	790	2370
2400	405	1215

Baud rate	Calculated minimum USS_PORT call Interval (milliseconds)	Drive message interval timeout per drive (milliseconds)
4800	212.5	638
9600	116.3	349
19200	68.2	205
38400	44.1	133
57600	36.1	109
115200	28.1	85

12.4.2 USS_DRV instruction

Table 12- 35 USS_DRV instruction

LAD / FBD	SCL	Description
Default view  Expanded view 	"USS_DRV_DB"(RUN:=_bool_in_, OFF2:=_bool_in_, OFF3:=_bool_in_, F_ACK:=_bool_in_, DIR:=_bool_in_, DRIVE:=_usint_in_, PZD_LEN:=_usint_in_, SPEED_SP:=_real_in_, CTRL3:=_word_in_, CTRL4:=_word_in_, CTRL5:=_word_in_, CTRL6:=_word_in_, CTRL7:=_word_in_, CTRL8:=_word_in_, NDR=>_bool_out_, ERROR=>_bool_out_, STATUS=>_word_out_, RUN_EN=>_bool_out_, D_DIR=>_bool_out_, INHIBIT=>_bool_out_, FAULT=>_bool_out_, SPEED=>_real_out_, STATUS1=>_word_out_, STATUS3=>_word_out_, STATUS4=>_word_out_, STATUS5=>_word_out_, STATUS6=>_word_out_, STATUS7=>_word_out_, STATUS8=>_word_out_);	The USS_DRV instruction exchanges data with a drive by creating request messages and interpreting the drive response messages. A separate function block should be used for each drive, but all USS functions associated with one USS network and PtP communication port must use the same instance data block. You must create the DB name when you place the first USS_DRV instruction and then reference the DB that was created by the initial instruction usage. STEP 7 automatically creates the DB when you insert the instruction.

[1] LAD and FBD: Expand the box to reveal all the parameters by clicking the bottom of the box. The parameter pins that are grayed are optional and parameter assignment is not required.

Table 12- 36 Data types for the parameters

Parameter and type		Data type	Description
RUN	IN	Bool	Drive start bit: When true, this input enables the drive to run at the preset speed. When RUN goes to false while a drive is running, the motor will be ramped down to a stop. This behavior differs from the dropping power (OFF2) or braking the motor (OFF3).
OFF2	IN	Bool	Electrical stop bit: When false, this bit cause the drive to coast to a stop with no braking.
OFF3	IN	Bool	Fast stop bit: When false, this bit causes a fast stop by braking the drive rather than just allowing the drive to coast to a stop.
F_ACK	IN	Bool	Fault acknowledge bit: This bit is set to reset the fault bit on a drive. The bit is set after the fault is cleared to indicate to the drive it no longer needs to indicate the previous fault.
DIR	IN	Bool	Drive direction control: This bit is set to indicate that the direction is forward (for positive SPEED_SP).
DRIVE	IN	USInt	Drive address: This input is the address of the USS drive. The valid range is drive 1 to drive 16.
PZD_LEN	IN	USInt	Word length: This is the number of words of PZD data. The valid values are 2, 4, 6, or 8 words. The default value is 2.
SPEED_SP	IN	Real	Speed set point: This is the speed of the drive as a percentage of configured frequency. A positive value specifies forward direction (when DIR is true). Valid range is 200.00 to -200.00.
CTRL3	IN	Word	Control word 3: A value written to a user-configurable parameter on the drive. You must configure this on the drive. (optional parameter)
CTRL4	IN	Word	Control word 4: A value written to a user-configurable parameter on the drive. You must configure this on the drive. (optional parameter)
CTRL5	IN	Word	Control word 5: A value written to a user-configurable parameter on the drive. You must configure this on the drive. (optional parameter)
CTRL6	IN	Word	Control word 6: A value written to a user-configurable parameter on the drive. You must configure this on the drive. (optional parameter)
CTRL7	IN	Word	Control word 7: A value written to a user-configurable parameter on the drive. You must configure this on the drive. (optional parameter)
CTRL8	IN	Word	Control word 8: A value written to a user-configurable parameter on the drive. You must configure this on the drive. (optional parameter)
NDR	OUT	Bool	New data ready: When true, the bit indicates that the outputs contain data from a new communication request.
ERROR	OUT	Bool	Error occurred: When true, this indicates that an error has occurred and the STATUS output is valid. All other outputs are set to zero on an error. Communication errors are only reported on the USS_PORT instruction ERROR and STATUS outputs.
STATUS	OUT	Word	The status value of the request indicates the result of the scan. This is not a status word returned from the drive.
RUN_EN	OUT	Bool	Run enabled: This bit indicates whether the drive is running.
D_DIR	OUT	Bool	Drive direction: This bit indicates whether the drive is running forward.
INHIBIT	OUT	Bool	Drive inhibited: This bit indicates the state of the inhibit bit on the drive.
FAULT	OUT	Bool	Drive fault: This bit indicates that the drive has registered a fault. You must fix the problem and then set the F_ACK bit to clear this bit when set.

Parameter and type		Data type	Description
SPEED	OUT	Real	Drive Current Speed (scaled value of drive status word 2): The value of the speed of the drive as a percentage of configured speed.
STATUS1	OUT	Word	Drive Status Word 1: This value contains fixed status bits of a drive.
STATUS3	OUT	Word	Drive Status Word 3: This value contains a user-configurable status word on the drive.
STATUS4	OUT	Word	Drive Status Word 4: This value contains a user-configurable status word on the drive.
STATUS5	OUT	Word	Drive Status Word 5: This value contains a user-configurable status word on the drive.
STATUS6	OUT	Word	Drive Status Word 6: This value contains a user-configurable status word on the drive.
STATUS7	OUT	Word	Drive Status Word 7: This value contains a user-configurable status word on the drive.
STATUS8	OUT	Word	Drive Status Word 8: This value contains a user-configurable status word on the drive.

When the initial USS_DRV execution occurs, the drive indicated by the USS address (parameter DRIVE) is initialized in the Instance DB. After this initialization, subsequent executions of USS_PORT can begin communication to the drive at this drive number.

Changing the drive number requires a CPU STOP-to-RUN mode transition that initializes the instance DB. Input parameters are configured into the USS TX message buffer and outputs are read from a "previous" valid response buffer if any exists. There is no data transmission during USS_DRV execution. Drives communicate when USS_PORT is executed. USS_DRV only configures the messages to be sent and interprets data that might have been received from a previous request.

You can control the drive direction of rotation using either the DIR input (Bool) or using the sign (positive or negative) with the SPEED_SP input (Real). The following table indicates how these inputs work together to determine the drive direction, assuming the motor is wired for forward rotation.

Table 12- 37 Interaction of the SPEED_SP and DIR parameters

SPEED_SP	DIR	Drive rotation direction
Value > 0	0	Reverse
Value > 0	1	Forward
Value < 0	0	Forward
Value < 0	1	Reverse

12.4.3 USS_PORT instruction

Table 12- 38 USS_PORT instruction

LAD / FBD	SCL	Description

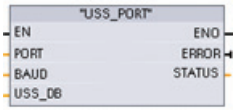
	`USS_PORT(`
	`PORT:=_uint_in_,`
	`BAUD:=_dint_in_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`USS_DB:=_fbtref_inout_);`
		The USS_PORT instruction handles communication over a USS network.

Table 12- 39 Data types for the parameters

Parameter and type		Data type	Description
PORT	IN	Port	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table.
BAUD	IN	DInt	The baud rate used for USS communication.
USS_DB	INOUT	USS_BASE	The name of the instance DB that is created and initialized when a USS_DRV instruction is placed in your program.
ERROR	OUT	Bool	When true, this output indicates that an error has occurred and the STATUS output is valid.
STATUS	OUT	Word	The status value of the request indicates the result of the scan or initialization. Additional information is available in the "USS_Extended_Error" variable for some status codes.

Typically, there is only one USS_PORT instruction per PtP communication port in the program, and each call of this function handles a transmission to or from a single drive. All USS functions associated with one USS network and PtP communication port must use the same instance DB.

Your program must execute the USS_PORT instruction often enough to prevent drive timeouts. USS_PORT is usually called from a cyclic interrupt OB to prevent drive timeouts and keep the most recent USS data updates available for USS_DRV calls.

12.4.4 USS_RPM instruction

Table 12- 40 USS_RPM instruction

LAD / FBD	SCL	Description

	`USS_RPM(REQ:=_bool_in_,`
	`DRIVE:=_usint_in_,`
	`PARAM:=_uint_in_,`
	`INDEX:=_uint_in_,`
	`DONE=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`VALUE=>_variant_out_,`
	`USS_DB:=_fbtref_inout_);`
		The USS_RPM instruction reads a parameter from a drive. All USS functions associated with one USS network and PtP communication port must use the same data block. USS_RPM must be called from a main program cycle OB.

Table 12- 41 Data types for the parameters

Parameter type		Data type	Description
REQ	IN	Bool	Send request: When true, REQ indicates that a new read request is desired. This is ignored if the request for this parameter is already pending.
DRIVE	IN	USInt	Drive address: DRIVE is the address of the USS drive. The valid range is drive 1 to drive 16.
PARAM	IN	UInt	Parameter number: PARAM designates which drive parameter is written. The range of this parameter is 0 to 2047. On some drives, the most significant byte can access PARAM values greater than 2047. See your drive manual for details on how to access an extended range.
INDEX	IN	UInt	Parameter index: INDEX designates which Drive Parameter index is to be written. A 16-bit value where the Least Significant Byte is the actual index value with a range of (0 to 255). The Most Significant Byte may also be used by the drive and is drive-specific. See your drive manual for details.
USS_DB	INOUT	USS_BASE	Then name of the instance DB that is created and initialized when a USS_DRV instruction is placed in your program.
VALUE	IN	Word, Int, UInt, DWord, DInt, UDInt, Real	This is the value of the parameter that was read and is valid only when the DONE bit is true.
DONE[1]	OUT	Bool	When true, indicates that the VALUE output holds the previously requested read parameter value. This bit is set when USS_DRV sees the read response data from the drive. This bit is reset when either: you request the response data via another USS_RPM poll, or on the second of the next two calls to USS_DRV

Parameter type		Data type	Description
ERROR	OUT	Bool	Error occurred: When true, ERROR indicates that an error has occurred and the STATUS output is valid. All other outputs are set to zero on an error. Communication errors are only reported on the USS_PORT instruction ERROR and STATUS outputs.
STATUS	OUT	Word	STATUS indicates the result of the read request. Additional information is available in the "USS_Extended_Error" variable for some status codes.

[1] The DONE bit indicates that valid data has been read from the referenced motor drive and delivered to the CPU. It does not indicate that the USS library is capable of immediately reading another parameter. A blank PKW request must be sent to the motor drive and must also be acknowledged by the instruction before the parameter channel for the specific drive becomes available for use. Immediately calling a USS_RPM or USS_WPM FC for the specified motor drive will result in a 0x818A error.

12.4.5 USS_WPM instruction

Note

EEPROM write operations (for the EEPROM inside a USS drive)

Do not overuse the EEPROM permanent write operation. Minimize the number of EEPROM write operations to extend the EEPROM life.

Table 12- 42 USS_WPM instruction

LAD / FBD	SCL	Description

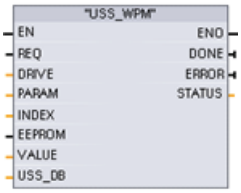
	`USS_WPM(REQ:=_bool_in_,`
	`DRIVE:=_usint_in_,`
	`PARAM:=_uint_in_,`
	`INDEX:=_uint_in_,`
	`EEPROM:=_bool_in_,`
	`VALUE:=_variant_in_,`
	`DONE=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`USS_DB:=_fbtref_inout_);`
		The USS_WPM instruction modifies a parameter in the drive. All USS functions associated with one USS network and PtP communication port must use the same data block.
		USS_WPM must be called from a main program cycle OB.

Table 12- 43 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	Send request: When true, REQ indicates that a new write request is desired. This is ignored if the request for this parameter is already pending.
DRIVE	IN	USInt	Drive address: DRIVE is the address of the USS drive. The valid range is drive 1 to drive 16.

Parameter and type		Data type	Description
PARAM	IN	UInt	Parameter number: PARAM designates which drive parameter is written. The range of this parameter is 0 to 2047. On some drives, the most significant byte can access PARAM values greater than 2047. See your drive manual for details on how to access an extended range.
INDEX	IN	UInt	Parameter index: INDEX designates which Drive Parameter index is to be written. A 16-bit value where the least significant byte is the actual index value with a range of (0 to 255). The most significant byte may also be used by the drive and is drive-specific. See your drive manual for details.
EEPROM	IN	Bool	Store To Drive EEPROM: When true, a write drive parameter transaction will be stored in the drive EEPROM. If false, the write is temporary and will not be retained if the drive is power cycled.
VALUE	IN	Word, Int, UInt, DWord, DInt, UDInt, Real	The value of the parameter that is to be written. It must be valid on the transition of REQ.
USS_DB	INOUT	USS_BASE	The name of the instance DB that is created and initialized when a USS_DRV instruction is placed in your program.
DONE[1]	OUT	Bool	When true, DONE indicates that the input VALUE has been written to the drive. This bit is set when USS_DRV sees the write response data from the drive. This bit is reset when either you request the response data via another USS_RPM poll, or on the second of the next two calls to USS_DRV
ERROR	OUT	Bool	When true, ERROR indicates that an error has occurred and the STATUS output is valid. All other outputs are set to zero on an error. Communication errors are only reported on the USS_PORT instruction ERROR and STATUS outputs.
STATUS	OUT	Word	STATUS indicates the result of the write request. Additional information is available in the "USS_Extended_Error" variable for some status codes.

[1] The DONE bit indicates that valid data has been read from the referenced motor drive and delivered to the CPU. It does not indicate that the USS library is capable of immediately reading another parameter. A blank PKW request must be sent to the motor drive and must also be acknowledged by the instruction before the parameter channel for the specific drive becomes available for use. Immediately calling a USS_RPM or USS_WPM FC for the specified motor drive will result in a 0x818A error.

12.4.6 USS status codes

USS instruction status codes are returned at the STATUS output of the USS functions.

Table 12- 44 STATUS codes [1]

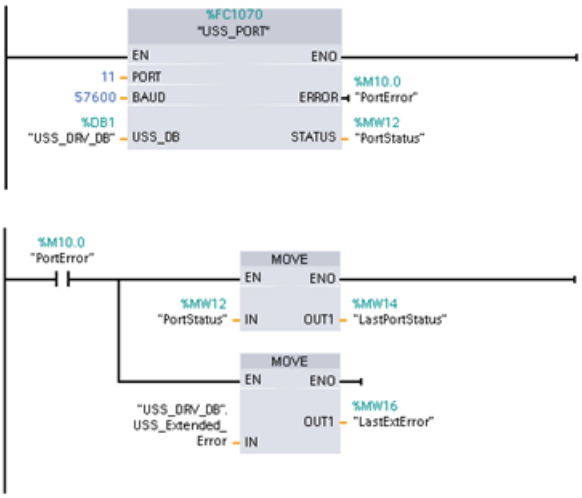
STATUS (W#16#....)	Description
0000	No error
8180	The length of the drive response did not match the characters received from the drive. The drive number where the error occurred is returned in the "USS_Extended_Error" variable. See the extended error description below this table.
8181	VALUE parameter was not a Word, Real or DWord data type.

STATUS (W#16#....)	Description
8182	The user supplied a Word for a parameter value and received a DWord or Real from the drive in the response.
8183	The user supplied a DWord or Real for a parameter value and received a Word from the drive in the response.
8184	The response telegram from drive had a bad checksum. The drive number where the error occurred is returned in the "USS_Extended_Error" variable. See the extended error description below this table.
8185	Illegal drive address (valid drive address range: 1 to16)
8186	The speed set point is out of the valid range (valid speed SP range: -200% to 200%).
8187	The wrong drive number responded to the request sent. The drive number where the error occurred is returned in the "USS_Extended_Error" variable. See the extended error description below this table.
8188	Illegal PZD word length specified (valid range = 2, 4, 6 or 8 words)
8189	Illegal Baud Rate was specified.
818A	The parameter request channel is in use by another request for this drive.
818B	The drive has not responded to requests and retries. The drive number where the error occurred is returned in the "USS_Extended_Error" variable. See the extended error description below this table.
818C	The drive returned an extended error on a parameter request operation. See the extended error description below this table.
818D	The drive returned an illegal access error on a parameter request operation. See your drive manual for information of why parameter access may be limited.
818E	The drive has not been initialized. This error code is returned to USS_RPM or USS_WPM when USS_DRV, for that drive, has not been called at least once. This keeps the initialization on first scan of USS_DRV from overwriting a pending parameter read or write request, since it initializes the drive as a new entry. To fix this error, call USS_DRV for this drive number.
80Ax-80Fx	Specific errors returned from PtP communication FBs called by the USS Library - These error code values are not modified by the USS library and are defined in the PtP instruction descriptions.

[1] In addition to the USS instruction errors listed above, errors can be returned from the underlying PtP communication instructions (Page 532).

For several STATUS codes, additional information is provided in the "USS_Extended_Error" variable of the USS_DRV Instance DB. For STATUS codes hexadecimal 8180, 8184, 8187, and 818B, USS_Extended_Error contains the drive number where the communication error occurred. For STATUS code hexadecimal 818C, USS_Extended_Error contains a drive error code returned from the drive when using a USS_RPM or USS_WPM instruction.

Communication errors (STATUS = 16#818B) are only reported on the USS_PORT instruction and not on the USS_DRV instruction. For example, if the network is not properly terminated then it is possible for a drive to go to RUN but the USS_DRV instruction will show all 0's for the output parameters. In this case, you can only detect the communication error on the USS_PORT instruction. Since this error is only visible for one scan, you will need to add some capture logic as illustrated in the following example. In this example, when the error bit of the USS_PORT instruction is TRUE, then the STATUS and the USS_Extended_Error values are saved into M memory. The drive number is placed in USS_Extended_Error variable when the STATUS code value is hexadecimal 8180, 8184, 8187, or 818B.



Network 1 "PortStatus" port status and "USS_DRV_DB".USS_Extended_Error extended error code values are only valid for one program scan. The values must be captured for later processing.

Network 2 The "PortError" contact triggers the storage of the "PortStatus" value in "LastPortStatus" and the "USS_DRV_DB".USS_Extended_Error value in "LastExtError".

USS drives support read and write access to a drive's internal parameters. This feature allows remote control and configuration of the drive. Drive parameter access operations can fail due to errors such as values out of range or illegal requests for a drive's current mode. The drive generates an error code value that is returned in the "USS_Extended_Error" variable. This error code value is only valid for the last execution of a USS_RPM or USS_WPM instruction. The drive error code is put into USS_Extended_Error variable when the STATUS code value is hexadecimal 818C. The error code value of "USS_Extended_Error" depends on the drive model. See the drive's manual for a description of the extended error codes for read and write parameter operations.

12.4.7 General drive setup information

General drive setup requirements

- The drives must be set to use 4 PKW words.
- The drives can be configured for 2, 4, 6, or 8 PZD words.
- The number of PZD word's in the drive must match PZD_LEN input on the USS_DRV instruction for that drive.
- The baud rate in all the drives must match the BAUD input on the USS_PORT instruction.
- The drive must be set for remote control.
- The drive must be set for frequency set-point to USS on COM Link.
- The drive address must be set to 1 to 16 and match the DRIVE input on the USS_DRV block for that drive.
- The drive direction control must be set to use the polarity of the drive set-point.
- The RS485 network must be terminated properly.

Connecting a MicroMaster drive

This information about SIEMENS MicroMaster drives is provided as an example. For other drives, refer to the drive's manual for setup instructions.

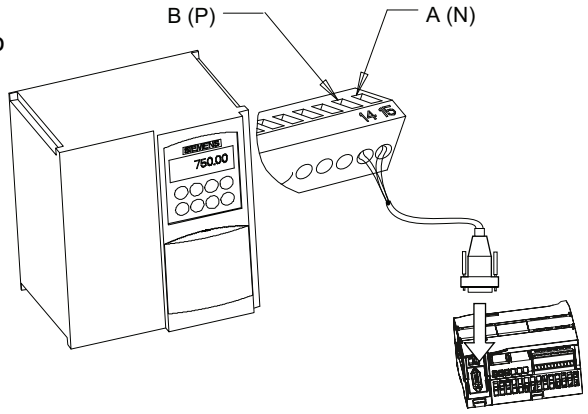
To make the connection to a MicroMaster Series 4 (MM4) drive, insert the ends of the RS-485 cable into the two caged-clamp, screw-less terminals provided for USS operation. Standard PROFIBUS cable and connectors can be used to connect the S7-1200.

⚠ CAUTION

Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable

These unwanted currents can cause communications errors or damage equipment. Be sure all equipment that you are about to connect with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows. The shield must be tied to chassis ground or pin 1 on the 9-pin connector. It is recommended that you tie wiring terminal 2--0V on the MicroMaster drive to chassis ground.

The two wires at the opposite end of the RS-485 cable must be inserted into the MM4 drive terminal blocks. To make the cable connection on a MM4 drive, remove the drive cover(s) to access the terminal blocks. See the MM4 user manual for details about how to remove the covers(s) of your specific drive.



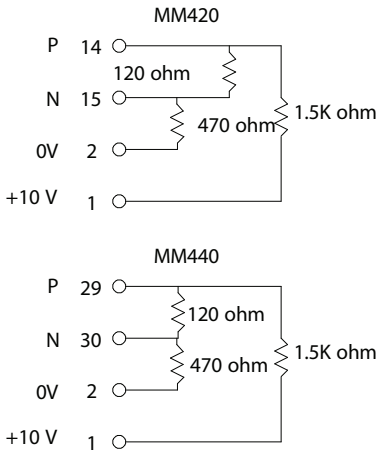
The terminal block connections are labeled numerically. Using a PROFIBUS connector on the S7-1200 side, connect the A terminal of the cable to the drive terminal 15 (for an MM420) or terminal 30 (MM440). Connect the B terminal of B (P) A (N) the cable connector to terminal 14 (MM420) or terminal 29 (MM440).

If the S7-1200 is a terminating node in the network, or if the connection is point-to-point, it is necessary to use terminals A1 and B1 (not A2 and B2) of the connector since they allow the termination settings to be set (for example, with DP connector type 6ES7 972--0BA40--0X40).

⚠ CAUTION

Make sure the drive covers are replaced properly before supplying power to the unit.

If the drive is configured as the terminating node in the network, then termination and bias resistors must also be wired to the appropriate terminal connections. This diagram shows examples of the MM4 drive connections necessary for termination and bias.



Setting up the MM4 drive

Before you connect a drive to the S7-1200, you must ensure that the drive has the following system parameters. Use the keypad on the drive to set the parameters:

1. Reset the drive to factory settings (optional).	P0010=30 P0970=1
If you skip step 1, then ensure that these parameters are set to the indicated values.	USS PZD length = P2012 Index 0=(2, 4, 6, or 8) USS PKW length = P2013 Index 0=4
2. Enable the read/write access to all parameters (Expert mode).	P0003=3
3. Check the motor settings for your drive. The settings will vary according to the motor(s) being used. To set the parameters P304, P305, P307, P310, and P311, you must first set parameter P010 to 1 (quick commissioning mode). When you are finished setting the parameters, set parameter P010 to 0. Parameters P304, P305, P307, P310, and P311 can only be changed in the quick commissioning mode.	P0304 = Rated motor voltage (V) P0305 = Rated motor current (A) P0307 = Rated motor power (W) P03010 = Rated motor frequency (Hz) P0311 = Rated motor speed
4. Set the local/remote control mode.	P0700 Index 0=5
5. Set selection of frequency set-point to USS on COM link.	P1000 Index 0=5
6. Ramp up time (optional) This is the time in seconds that it takes the motor to accelerate to maximum frequency.	P1120=(0 to 650.00)
7. Ramp down time (optional) This the time in seconds that it takes the motor to decelerate to a complete stop.	P1121=(0 to 650.00)
8. Set the serial link reference frequency:	P2000=(1 to 650 Hz)
9. Set the USS normalization:	P2009 Index 0=0

10. Set the baud rate of the RS-485 serial interface:	P2010 Index 0= 4 (2400 baud) 5 (4800 baud) 6 (9600 baud) 7 (19200 baud 8 (38400 baud) 9 (57600 baud) 12 (115200 baud)
11. Enter the Slave address. Each drive (a maximum of 31) can be operated over the bus.	P2011 Index 0=(0 to 31)
12. Set the serial link timeout. This is the maximum permissible period between two incoming data telegrams. This feature is used to turn off the inverter in the event of a communications failure. Timing starts after a valid data telegram has been received. If a further data telegram is not received within the specified time period, the inverter will trip and display fault code F0070. Setting the value to zero switches off the control.	P2014 Index 0=(0 to 65,535 ms) 0=timeout disabled
13. Transfer the data from RAM to EEPROM:	P0971=1 (Start transfer) Save the changes to the parameter settings to EEPROM

12.5 Modbus communication

12.5.1 Overview of Modbus RTU and TCP communication

Modbus function codes

- A CPU operating as a Modbus RTU master (or Modbus TCP client) can read/write both data and I/O states in a remote Modbus RTU slave (or Modbus TCP server). Remote data can be read and processed in the user program.
- A CPU operating as a Modbus RTU slave (or Modbus TCP server) allows a supervisory device to read/write both data and I/O states in a remote CPU. The supervisor device can write new values in remote CPU memory that can be processed in the user program.

Table 12- 45 Read data functions: Read remote I/O and program data

Modbus function code	Read slave (server) functions - standard addressing
01	Read output bits: 1 to 2000 bits per request
02	Read input bits: 1 to 2000 bits per request
03	Read Holding registers: 1 to 125 words per request
04	Read input words: 1 to 125 words per request

Table 12- 46 Write data functions: Write remote I/O and modify program data

Modbus function code	Write slave (server) functions - standard addressing
05	Write one output bit: 1 bit per request
06	Write one holding register: 1 word per request
15	Write one or more output bits: 1 to 1968 bits per request
16	Write one or more holding registers: 1 to 123 words per request

- Modbus function codes 08 and 11 provide slave device communication diagnostic information.
- Modbus function code 0 broadcasts a message to all slaves (with no slave response). The broadcast function is not available for Modbus TCP, because communication is connection based.

Table 12- 47 Modbus network station addresses

Station		Address
RTU station	Standard station address	1 to 247
	Extended station address	1 to 65535
TCP station	Station address	IP address and port number

Modbus memory addresses

The actual number of Modbus memory addresses available depends on the CPU model, how much work memory exists, and how much CPU memory is used by other program data. The table below gives the nominal value of the address range.

Table 12- 48 Modbus memory addresses

Station		Address range
RTU station	Standard memory address	10K
	Extended memory address	64K
TCP station	Standard memory address	10K

Modbus RTU communication

Modbus RTU (Remote Terminal Unit) is a standard network communication protocol that uses the RS232 or RS485 electrical connection for serial data transfer between Modbus network devices. You can add PtP (Point to Point) network ports to a CPU with a RS232 or RS485 CM or a RS485 CB.

Modbus RTU uses a master/slave network where all communications are initiated by a single Master device and slaves can only respond to a master's request. The master sends a request to one slave address and only that slave address responds to the command.

Modbus TCP communication

Modbus TCP (Transmission Control Protocol) is a standard network communication protocol that uses the PROFINET connector on the CPU for TCP/IP communication. No additional communication hardware module is required.

Modbus TCP uses Open User Communications (OUC) connections as a Modbus communication path. Multiple client-server connections may exist, in addition to the connection between STEP 7 and the CPU. Mixed client and server connections are supported up to the maximum number of connections allowed by the CPU model (Page 394).

Each MB_SERVER connection must use a unique instance DB and IP port number. Only 1 connection per IP port is supported. Each MB_SERVER (with its unique instance DB and IP port) must be executed individually for each connection.

Note

Modbus TCP will only operate correctly with CPU firmware release V1.02 or later. An attempt to execute the Modbus instructions on an earlier firmware version will result in an error.

A Modbus TCP client (master) must control the client-server connection with the DISCONNECT parameter. The basic Modbus client actions are shown below.

1. Initiate a connection to a particular server (slave) IP address and IP port number
2. Initiate client transmission of a Modbus messages and receive the server responses
3. When desired, initiate the disconnection of client and server to enable connection with a different server.

Modbus RTU instructions in your program

- MB_COMM_LOAD: One execution of MB_COMM_LOAD is used to set up PtP port parameters like baud rate, parity, and flow control. After a CPU port is configured for the Modbus RTU protocol, it can only be used by either the MB_MASTER or MB_SLAVE instructions.
- MB_MASTER: The Modbus master instruction enables the CPU to act as a Modbus RTU master device and communicate with one or more Modbus slave devices.
- MB_SLAVE: The Modbus slave instruction enables the CPU to act as a Modbus RTU slave device and communicate with a Modbus master device.

Modbus TCP instructions in your program

- MB_CLIENT: Make client-server TCP connection, send command message, receive response, and control the disconnection from the server
- MB_SERVER: Connect to a Modbus TCP client upon request, receive Modbus message, and send response

12.5.2 Modbus TCP

12.5.2.1 MB_CLIENT (Modbus TCP)

Table 12- 49 MB_CLIENT instruction

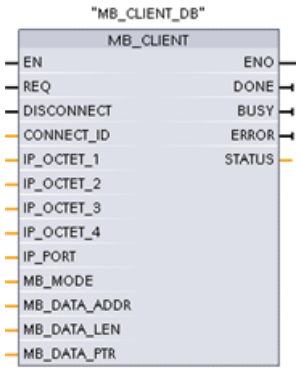
LAD / FBD	SCL	Description
	"MB_CLIENT_DB"(REQ:=_bool_in_, DISCONNECT:=_bool_in_, CONNECT_ID=_uint_in_, IP_OCTET_1:=_byte_in_, IP_OCTET_2:=_byte_in_, IP_OCTET_3:=_byte_in_, IP_OCTET_4:=_byte_in_, IP_PORT:=_uint_in_, MB_MODE:=_usint_in_, MB_DATA_ADDR:=_udint_in_, MB_DATA_LEN:=_uint_in_, DONE=>_bool_out_, BUSY=>_bool_out_, ERROR=>_bool_out_, STATUS=>_word_out_, MB_DATA_PTR:=_variant_inout_);	MB_CLIENT communicates as a Modbus TCP client through the PROFINET connector on the S7-1200 CPU. No additional communication hardware module is required. MB_CLIENT can make a client-server connection, send a Modbus function request, receive a response, and control the disconnection from a Modbus TCP server.

Table 12- 50 Data types for the parameters

Parameter and type		Data type	Description
REQ	In	Bool	FALSE = No Modbus communication request TRUE = Request to communicate with a Modbus TCP server
DISCONNECT	IN	Bool	The DISCONNECT parameter allows your program to control connection and disconnection with a Modbus server device. If DISCONNECT = 0 and a connection does not exist, then MB_CLIENT attempts to make a connection to the assigned IP address and port number. If DISCONNECT = 1 and a connection exists, then a disconnect operation is attempted. Whenever this input is enabled, no other operation will be attempted.
CONNECT_ID	IN	UInt	The CONNECT_ID parameter must uniquely identify each connection within the PLC. Each unique instance of the MB_CLIENT or MB_SERVER instruction must contain a unique CONNECT_ID parameter.
IP_OCTET_1	IN	USInt	Modbus TCP server IP address: Octet 1 8-bit part of the 32-bit IPv4 IP address of the Modbus TCPserver to which the client will connect and communicate using the Modbus TCP protocol.
IP_OCTET_2	IN	USInt	Modbus TCP server IP address: Octet 2
IP_OCTET_3	IN	USInt	Modbus TCP server IP address: Octet 3
IP_OCTET_4	IN	USInt	Modbus TCP server IP address: Octet 4
IP_PORT	IN	UInt	Default value = 502: The IP port number of the server to which the client will attempt to connect and ultimately communicate using the TCP/IP protocol.

Parameter and type		Data type	Description
MB_MODE	IN	USInt	Mode Selection: Assigns the type of request (read, write, or diagnostic). See the Modbus functions table below for details.
MB_DATA_ADDR	IN	UDInt	Modbus starting Address: Assigns the starting address of the data to be accessed by MB_CLIENT. See the Modbus functions table below for valid addresses.
MB_DATA_LEN	IN	UInt	Modbus data Length: Assigns the number of bits or words to be accessed in this request. See the Modbus functions table below for valid lengths
MB_DATA_PTR	IN_OUT	Variant	Pointer to the Modbus data register: The register buffers data going to or coming from a Modbus server. The pointer must assign a standard global DB or a M memory address.
DONE	OUT	Bool	The DONE bit is TRUE for one scan, after the last request was completed with no error.
BUSY	OUT	Bool	• 0 - No MB_CLIENT operation in progress • 1 - MB_CLIENT operation in progress
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the MB_CLIENT execution was terminated with an error. The error code value at the STATUS parameter is valid only during the single cycle where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code

REQ parameter

FALSE = No Modbus communication request
TRUE = Request to communicate with a Modbus TCP server

If no instance of MB_CLIENT is active and parameter DISCONNECT=0, when REQ=1 a new Modbus request will start. If the connection is not already established then a new connection will be made.

If the same instance of MB_CLIENT is executed again with DISCONNECT=0 and REQ=1, before the completion of the current request, then no subsequent Modbus transmission will be made. However, as soon as the current request is completed, a new request can be processed if MB_CLIENT is executed with REQ=1.

When the current MB_CLIENT communication request is complete, the DONE bit is TRUE for one cycle. The DONE bit can be used as a time gate to sequence multiple MB_CLIENT requests.

Note

Input data consistency during MB_CLIENT processing

Once a Modbus client initiates a Modbus operation, all the input states are saved internally and are then compared on each successive call. The comparison is used to determine if this particular call was the originator of the active client request. More than one MB_CLIENT call can be performed using a common instance DB.

As a result, it is important that the inputs are not changed during the period of time that a MB_CLIENT operation is actively being processed. If this rule is not followed, then a MB_CLIENT cannot determine that it is the active instance.

MB_MODE and MB_DATA_ADDR parameters select the Modbus communication function

MB_DATA_ADDR assigns the starting Modbus address of the data to be accessed. The MB_CLIENT instruction uses a MB_MODE input rather than a function code input.

The combination of MB_MODE and MB_DATA_ADDR values determine the function code that is used in the actual Modbus message. The following table shows the correspondence between parameter MB_MODE, Modbus function, and Modbus address range.

Table 12- 51 Modbus functions

MB_MODE	Modbus function	Data length	Operation and data	MB_DATA_ADDR
0	01	1 to 2000	Read output bits: 1 to 2000 bits per request	1 to 9999
0	02	1 to 2000	Read input bits: 1 to 2000 bits per request	10001 to 19999
0	03	1 to 125	Read Holding registers: 1 to 125 words per request	40001 to 49999 or 400001 to 465535
0	04	1 to 125	Read input words: 1 to 125 words per request	30001 to 39999
1	05	1	Write one output bit: One bit per request	1 to 9999
1	06	1	Write one holding register: 1 word per request	40001 to 49999 or 400001 to 465535
1	15	2 to 1968	Write multiple output bits: 2 to 1968 bits per request	1 to 9999
1	16	2 to 123	Write multiple holding registers: 2 to 123 words per request	40001 to 49999 or 400001 to 465535
2	15	1 to 1968	Write one or more output bits: 1 to 1968 bits per request	1 to 9999
2	16	1 to 123	Write one or more holding registers: 1 to 123 words per request	40001 to 49999 or 400001 to 465535
11	11	0	Read the server communication status word and event counter. The status word indicates busy (0 – not busy, 0xFFFF - busy). The event counter is incremented for each successful completion of a message. Both the MB_DATA_ADDR and MB_DATA_LEN parameters of MB_CLIENT are ignored for this function.	
80	08	1	Check server status using data diagnostic code 0x0000 (Loopback test – server echoes the request) 1 word per request	

MB_MODE	Modbus function	Data length	Operation and data	MB_DATA_ADDR
81	08	1	Reset server event counter using data diagnostic code 0x000A 1 word per request	
3 to 10, 12 to 79, 82 to 255			Reserved	

Note

MB_DATA_PTR assigns a buffer to store data read/written to/from a Modbus TCP server

The data buffer can be in a standard global DB or M memory address.

For a buffer in M memory, use the standard Any Pointer format. This is in the format P#"Bit Address" "Data Type" "Length", an example would be P#M1000.0 WORD 500.

MB_DATA_PTR assigns a communication buffer

- MB_CLIENT communication functions:
  - Read and write 1-bit data from Modbus server addresses (00001 to 09999)
  - Read 1-bit data from Modbus server addresses (10001 to 19999)
  - Read 16-bit word data from Modbus server addresses (30001 to 39999) and (40001 to 49999)
  - Write 16-bit word data to Modbus server addresses (40001 to 49999)
- Word or bit sized data is transferred to/from the DB or M memory buffer assigned by MB_DATA_PTR.
- If a DB is assigned as the buffer by MB_DATA_PTR, then you must assign data types to all DB data elements.
  - The 1-bit Bool data type represents one Modbus bit address
  - 16-bit single word data types like WORD, UInt, and Int represent one Modbus word address
  - 32-bit double word data types like DWORD, DInt, and Real represent two Modbus word addresses

- Complex DB elements can be assigned by MB_DATA_PTR, such as
  - Standard arrays
  - Named structures where each element is unique.
  - Named complex structures where each element has a unique name and a 16 or 32 bit data type.
- There is no requirement that the MB_DATA_PTR data areas be in the same global data block (or M memory area). You can assign one data block for Modbus reads, another data block for Modbus writes, or one data block for each MB_CLIENT station.

Multiple client connections

A Modbus TCP client can support concurrent connections up to the maximum number of Open User Communications connections allowed by the PLC. The total number of connections for a PLC, including Modbus TCP Clients and Servers, must not exceed the maximum number of supported Open User Communications connections (Page 394). The Modbus TCP connections may be shared between Client and/or Server type connections.

Individual client connections must follow these rules:

- Each MB_CLIENT connection must use a distinct instance DB
- Each MB_CLIENT connection must specify a unique server IP address
- Each MB_CLIENT connection must specify a unique connection ID
- Unique IP port numbers may or may not be required depending upon the server configuration

The Connection ID must be unique for each individual connection. This means a single, unique Connection ID must only be used with each individual instance DB. In summary, the instance DB and the Connection ID are paired together and must be unique for every connection.

Table 12- 52 MB_CLIENT user accessible static variables

Variable	Data type	Default	description
Blocked_Proc_Timeout	Real	3.0	Amount of time (in seconds) to wait upon a blocked Modbus client instance before removing this instance as being ACTIVE. This can occur, for example, when a client request has been issued and then application stops executing the client function before it has completely finished the request. The maximum S7-1200 limit is 55 seconds.
MB_Unit_ID	Word	255	The Modbus TCP protocol unit ID. This value is initialized to a default value of 65535. This field corresponds to the slave address field in the Modbus RTU protocol. The value should be changed if a Modbus TCP server is capable of being used as a gateway and is controlled by the corresponding application program within the Modbus server. Some devices require this value to be set to a value of 1.
RCV_TIMEOUT	Real	2.0	Time in seconds that the MB_CLIENT waits for a server to respond to a request.
Connected	Bool	0	Indicates whether the connection to the assigned server is connected or disconnected: 1=connected, 0=disconnected

Table 12- 53 MB_CLIENT protocol errors

STATUS (W#16#)	Response code to Modbus client (B#16#)	Modbus protocol errors
8381	01	Function code not supported
8382	03	Data length error
8383	02	Data address error or access outside the bounds of the MB_HOLD_REG address area
8384	03	Data value error
8385	03	Data diagnostic code value not supported (function code 08)

Table 12- 54 MB_CLIENT execution condition codes [1]

STATUS (W#16#)	MB_CLIENT parameter errors
7001	MB_CLIENT is waiting for a Modbus server response to a connect or disconnect request, on the assigned TCP port. This is only reported for the first execution of a connect or disconnect operation.
7002	MB_CLIENT is waiting for a Modbus server response to a connect or disconnect request, for the assigned TCP port. This will be reported for any subsequent executions, while waiting for completion of a connect or disconnect operation.
7003	A disconnect operation has successfully completed (Only valid for one PLC scan).
80C8	The server did not respond in the assigned time. MB_CLIENT must receive a response using the transaction ID that was originally transmitted within the assigned time or this error is returned. Check the connection to the Modbus server device. This error is only reported after any configured retries (if applicable) have been attempted.
8188	Invalid mode value
8189	Invalid data address value
818A	Invalid data length value
818B	Invalid pointer to the DATA_PTR area. This can be the combination of MB_DATA_ADDRESS + MB_DATA_LEN.
818C	Pointer to a optimized DATA_PTR area (must be a standard DB area or M memory area)
8200	The port is busy processing an existing Modbus request.
8380	Received Modbus frame is malformed or too few bytes have been received.
8387	The assigned Connection ID parameter is different from the ID used for previous requests. There can only be a single Connection ID used within each MB_CLIENT instance DB. This is also used as an internal error if the Modbus TCP protocol ID received from a server is not 0.
8388	A Modbus server returned a quantity of data that is different than what was requested. This applies to Modbus functions 15 or 16 only.

[1] In addition to the MB_CLIENT errors listed above, errors can be returned from the underlying T block communication instructions (TCON, TDISCON, TSEND, and TRCV).

See also

TCON, TDISCON, TSEND, AND TRCV (Page 408)

12.5.2.2 MB_SERVER (Modbus TCP)

Table 12- 55 MB_SERVER instruction

LAD / FBD	SCL	Description

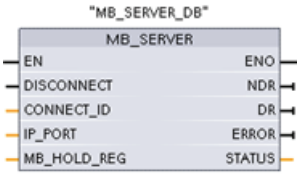
	`"MB_SERVER_DB"(`
	`DISCONNECT:=_bool_in_,`
	`CONNECT_ID:=_uint_in_,`
	`IP_PORT:=_uint_in_,`
	`NDR=>_bool_out_,`
	`DR=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`MB_HOLD_REG:=_variant_inout_);`
		MB_SERVER communicates as a Modbus TCP server through the PROFINET connector on the S7-1200 CPU. No additional communication hardware module is required.
		MB_SERVER can accept a request to connect with Modbus TCP client , receive a Modbus function request, and send a response message.

Table 12- 56 Data types for the parameters

Parameter and type		Data type	Description
DISCONNECT	IN	Bool	MB_SERVER attempts to make a "passive" connection with a partner device. This means that the server is passively listening for a TCP connection request from any requesting IP address. If DISCONNECT = 0 and a connection does not exist, then a passive connection can be initiated. If DISCONNECT = 1 and a connection exists, then a disconnect operation is initiated. This allows your program to control when a connection is accepted. Whenever this input is enabled, no other operation will be attempted.
CONNECT_ID	IN	UInt	CONNECT_ID uniquely identifies each connection within the PLC. Each unique instance of the MB_CLIENT or MB_SERVER instruction must contain a unique CONNECT_ID parameter.
IP_PORT	IN	UInt	Default value = 502: The IP port number that identifies the IP port that will be monitored for a connection request from a Modbus client. These TCP port numbers are not allowed for a MB_SERVER passive connection: 20, 21, 25, 80, 102, 123, 5001, 34962, 34963, and 34964.
MB_HOLD_REG	IN_OUT	Variant	Pointer to the MB_SERVER Modbus holding register: The holding register must either be a standard global DB or a M memory address. This memory area is used to hold the values a Modbus client is allowed to access using Modbus register functions 3 (read), 6 (write), and 16 (write).
NDR	OUT	Bool	New Data Ready: 0 = No new data, 1 = Indicates that new data has been written by a Modbus client
DR	OUT	Bool	Data Read: 0 = No data read, 1 = Indicates that data has been read by a Modbus client.
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after MB_SERVER execution was terminated with an error. The error code value at the STATUS parameter is valid only during the single cycle where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code

MB_SERVER allows incoming Modbus function codes (1, 2, 4, 5, and 15) to read or write bits and words directly in the input process image and output process image of the S7-1200 CPU. For data transfer function codes (3, 6, and 16), the MB_HOLD_REG parameter must be defined as a data type larger than a byte. The following table shows the mapping of Modbus addresses to the process image in the CPU.

Table 12- 57 Mapping of Modbus addresses to the process image

Modbus functions						S7-1200	
Codes	Function	Data area	Address range			Data area	CPU address
01	Read bits	Output	1	To	8192	Output Process Image	Q0.0 to Q1023.7
02	Read bits	Input	10001	To	18192	Input Process Image	I0.0 to I1023.7
04	Read words	Input	30001	To	30512	Input Process Image	IW0 to IW1022
05	Write bit	Output	1	To	8192	Output Process Image	Q0.0 to Q1023.7
15	Write bits	Output	1	To	8192	Output Process Image	Q0.0 to Q1023.7

Incoming Modbus message function codes function codes (3, 6, and 16) read or write words in a Modbus holding register which can be an M memory address range or a data block. The type of holding register is specified by the MB_HOLD_REG parameter.

Note

MB_HOLD_REG parameter assignment

The Modbus Holding Register can be in a standard global DB or a M memory address.

For A Modbus holding register in M memory, use the standard Any Pointer format. This is in the format P#"Bit Address" "Data Type" "Length". An example would be P#M1000.0 WORD 500

The following table shows examples of Modbus address to holding register mapping used for Modbus function codes 03 (read words), 06 (write word), and 16 (write words). The actual upper limit of DB addresses is determined by the maximum work memory limit and M memory limit, for each CPU model.

Table 12- 58 Mapping examples of Modbus address to CPU memory address

Modbus Address	MB_HOLD_REG parameter examples		
	P#M100.0 Word 5	P#DB10.DBx0.0 Word 5	"Recipe".ingredient
40001	MW100	DB10.DBW0	"Recipe".ingredient[1]
40002	MW102	DB10.DBW2	"Recipe".ingredient[2]
40003	MW104	DB10.DBW4	"Recipe".ingredient[3]
40004	MW106	DB10.DBW6	"Recipe".ingredient[4]
40005	MW108	DB10.DBW8	"Recipe".ingredient[5]

Multiple server connections

Multiple server connections may be created. This permits a single PLC to establish concurrent connections to multiple Modbus TCP clients.

A Modbus TCP server can support concurrent connections up to the maximum number of Open User Communications connections allowed by the PLC. The total number of connections for a PLC, including Modbus TCP Clients and Servers, must not exceed the maximum number of supported Open User Communications connections (Page 394). The Modbus TCP connections may be shared between Client and/or Server type connections.

Individual server connection must follow these rules:

- Each MB_SERVER connection must use a distinct instance DB.
- Each MB_SERVER connection must be established with a unique IP port number. Only 1 connection per port is supported.
- Each MB_SERVER connection must use a unique connection ID.
- The MB_SERVER must be called individually for each connection (with its respective instance DB).

The Connection ID must be unique for each individual connection. This means a single, unique Connection ID must only be used with each individual instance DB. In summary, the instance DB and the Connection ID are paired together and must be unique for every connection.

Table 12- 59 Modbus diagnostic function codes

MB_SERVER Modbus diagnostic functions		
Codes	Sub-function	Description
08	0x0000	Return query data echo test: The MB_SERVER will echo back to a Modbus client a word of data that is received.
08	0x000A	Clear communication event counter: The MB_SEVER will clear out the communication event counter that is used for Modbus function 11.
11		Get communication event counter: The MB_SERVER uses an internal communication event counter for recording the number of successful Modbus read and write requests that are sent to the Modbus server. The counter does not increment on any Function 8 or Function 11 requests. It is also not incremented on any requests that result in a communication error. The broadcast function is not available for Modbus TCP, because only one client-server connection exists at any one time.

MB_SERVER variables

This table shows the public static variables stored in the MB_SERVER instance data block that can be used in your program

Table 12- 60 MB_SERVER public static variables

Variable	Data type	Default value	Description
HR_Start_Offset	Word	0	Assigns the starting address of the Modbus Holding register
Request_Count	Word	0	The number of all requests received by this server.
Server_Message_Count	Word	0	The number of requests received for this specific server.
Xmt_Rcv_Count	Word	0	The number of transmissions or receptions that have encountered an error. Also, incremented if a message is received that is an invalid Modbus message.

Variable	Data type	Default value	Description
Exception_Count	Word	0	Modbus specific errors that require a returned exception
Success_Count	Word	0	The number of requests received for this specific server that have no protocol errors.
Connected	Bool	0	Indicates whether the connection to the assigned client is connected or disconnected: 1=connected, 0=disconnected

Your program can write values to the HR_Start_Offset and control Modbus server operations. The other variables can be read to monitor Modbus status.

HR_Start_Offset

Modbus holding register addresses begin at 40001 . These addresses correspond to the beginning PLC memory address of the holding register. However, you can configure the "HR_Start_Offset" variable to start the beginning Modbus holding register address at another value instead of 40001.

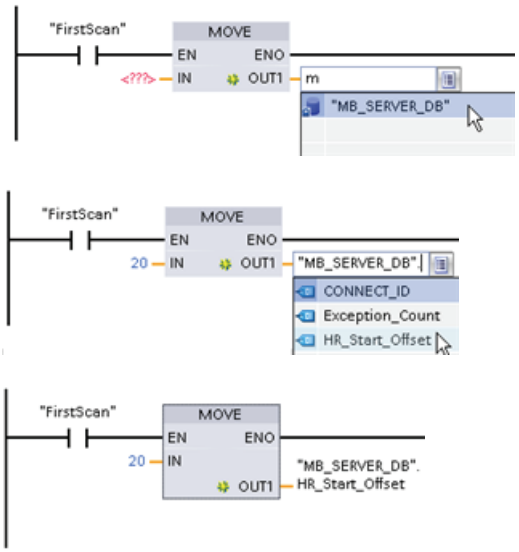
For example, if the holding register is configured to start at MW100 and is 100 words long. An offset of 20 specifies a beginning holding register address of 40021 instead of 40001. Any address below 40021 and above 40119 will result in an addressing error.

Table 12- 61 Example of Modbus holding register addressing

HR_Start_Offset	Address	Minimum	Maximum
0	Modbus address (Word)	40001	40099
	S7-1200 address	MW100	MW298
20	Modbus address (Word)	40021	40119
	S7-1200 address	MW100	MW298

HR_Start_Offset is a word value that specifies the starting address of the Modbus holding register and is stored in the MB_SERVER instance data block. You can set this public static variable value by using the parameter helper drop-list, after MB_SERVER is placed in your program.

For example, after MB_SERVER is placed in a LAD network, you can go to a previous network and assign the HR_Start_Offset value. The value must be assigned prior to execution of MB_SERVER.



Entering a Modbus server variable using the default DB name:

1. Set the cursor in the parameter field and type an m character.
2. Select "MB_SERVER_DB" from the drop-list of DB names.
3. Select "MB_SERVER_DB.HR_Start_Offset" from the drop-list of DB variables.

Table 12- 62 MB_SERVER execution condition codes [1]

STATUS (W#16#)	Response code to Modbus server (B#16#)	Modbus protocol errors
7001		MB_SERVER is waiting for a Modbus client to connect to the assigned TCP port. This code is reported on the first execution of a connect or disconnect operation.
7002		MB_SERVER is waiting for a Modbus client to connect to the assigned TCP port. This code is reported for any subsequent executions, while waiting for completion of a connect or disconnect operation.
7003		A disconnect operation has successfully completed (Only valid for one PLC scan).
8187		Invalid pointer to MB_HOLD_REG: area is too small
818C		Pointer to a optimized MB_HOLD_REG area (must be a standard DB area or M memory area) or Blocked process timeout exceeds the limit of 55 seconds. (S7-1200 specific)
8381	01	Function code not supported
8382	03	Data length error
8383	02	Data address error or access outside the bounds of the MB_HOLD_REG address area
8384	03	Data value error
8385	03	Data diagnostic code value not supported (function code 08)

[1] In addition to the MB_SERVER errors listed above, errors can be returned from the underlying T block communication instructions (TCON, TDISCON, TSEND, and TRCV).

See also

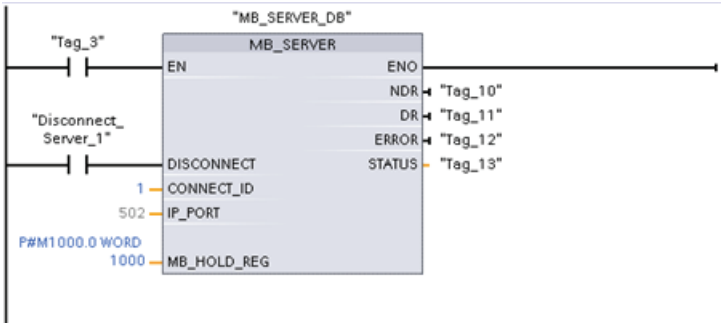
TCON, TDISCON, TSEND, AND TRCV (Page 408)

12.5.2.3 MB_SERVER example: Multiple TCP connections

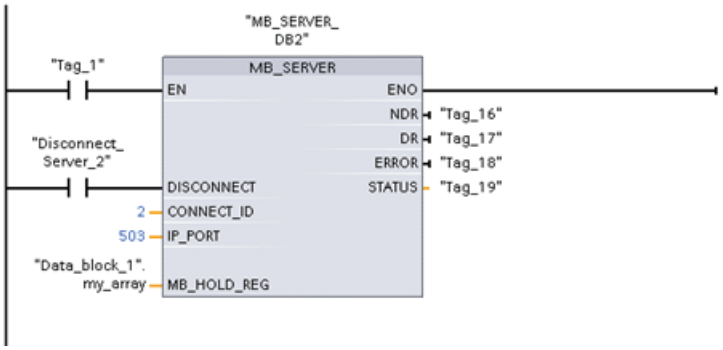
You can have multiple Modbus TCP server connections. To accomplish this, MB_SERVER must be independently executed for each connection. Each connection must use an independent instance DB, connection ID, and IP port. The S7-1200 allows only one connection per IP port.

For best performance, MB_SERVER should be executed every cycle for each connection.

Network 1: Connection #1 with independent IP_PORT, connection ID, and instance DB



Network 2: Connection #2 with independent IP_PORT, connection ID, and instance DB



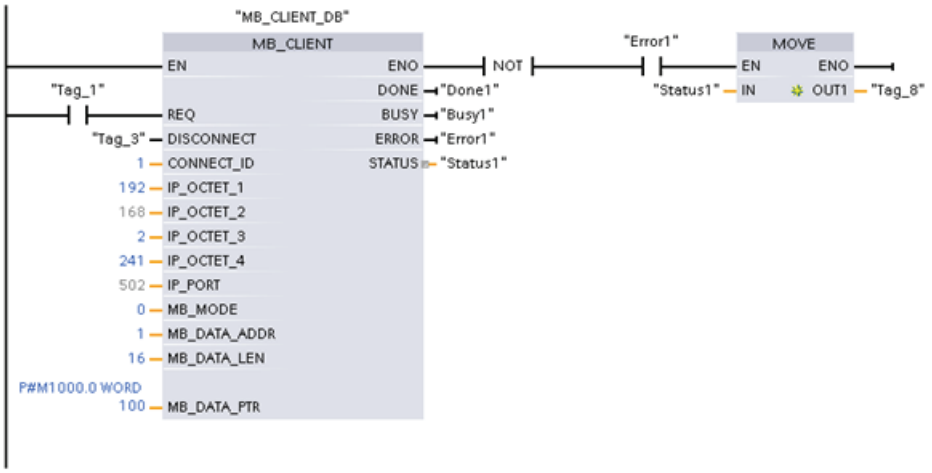
12.5.2.4 MB_CLIENT example 1: Multiple requests with common TCP connection

Multiple Modbus client requests can be sent over the same connection. To accomplish this, use the same instance DB, connection ID, and port number.

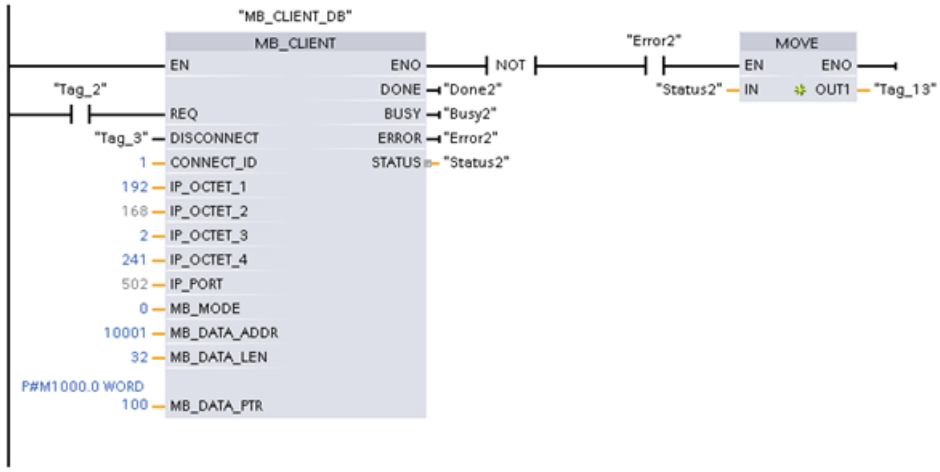
Only 1 client can be active at any given time. Once a client completes its execution, the next client begins execution. Your program is responsible for the order of execution.

The example shows both clients writing to the same memory area. Also, a returned error is captured which is optional.

Network 1: Modbus function 1 - Read 16 output image bits



Network 2: Modbus function 2 - Read 32 input image bits



12.5.2.5 MB_CLIENT example 2: Multiple requests with different TCP connections

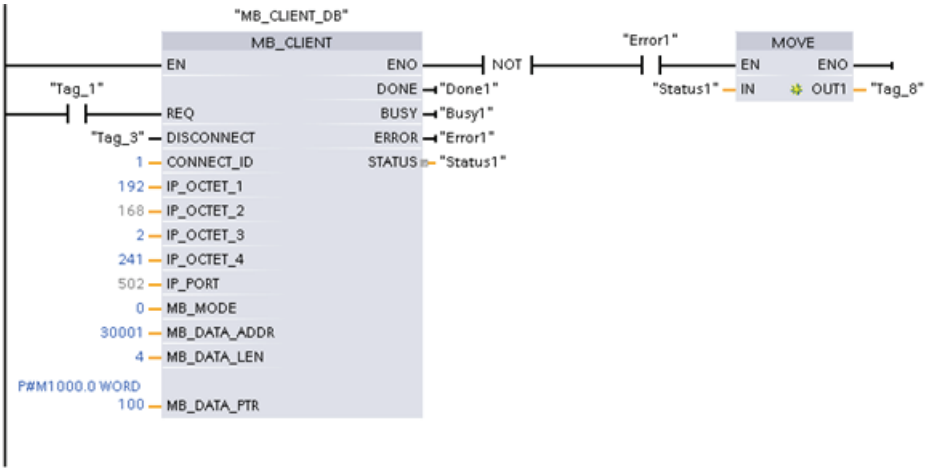
Modbus client requests can be sent over different connections. To accomplish this, different instance DBs, IP addresses, and connection IDs must be used.

The port number must be different if the connections are established to the same Modbus server. If the connections are on different servers, there is no port number restriction.

The example shows both clients writing to the same memory area. Also, a returned error is captured which is optional.

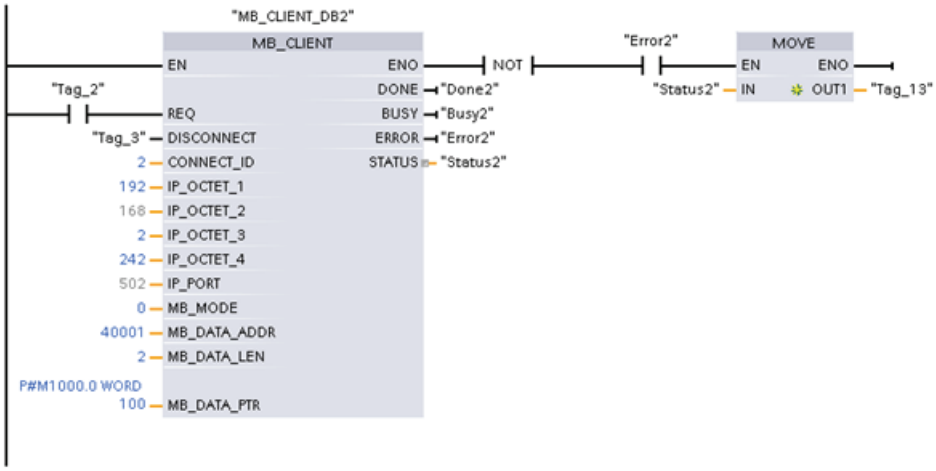
Network 1:

Modbus function 4 - Read input words (in S7-1200 memory)



Network 2:

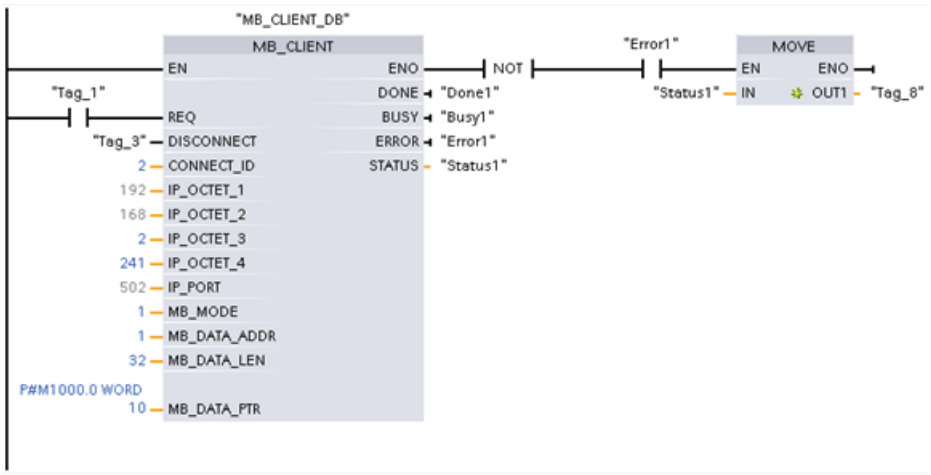
Modbus function 3 - Read holding register words (in S7-1200 memory)



12.5.2.6 MB_CLIENT example 3: Output image write request

This example shows a Modbus client request to write the S7-1200 output image.

Network 1: Modbus function 15 - Write S7-1200 output image bits

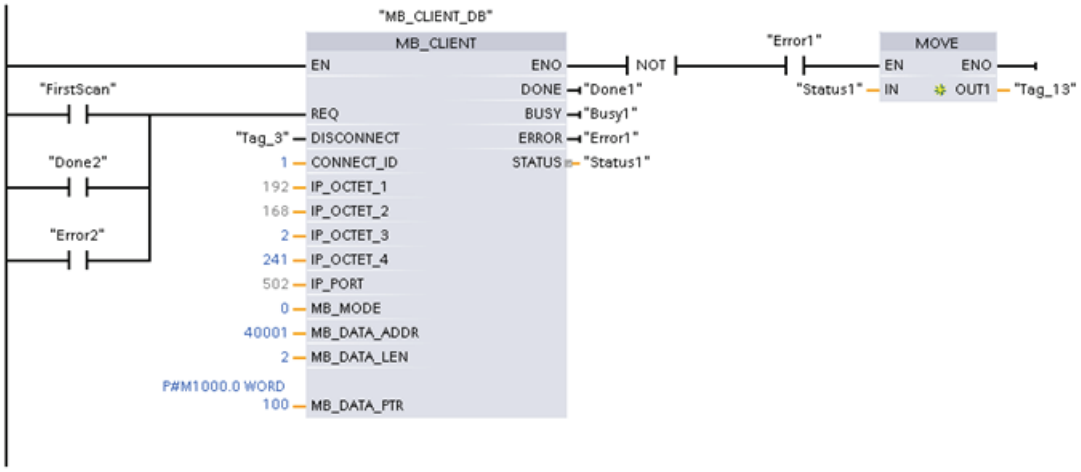


12.5.2.7 MB_CLIENT example 4: Coordinating multiple requests

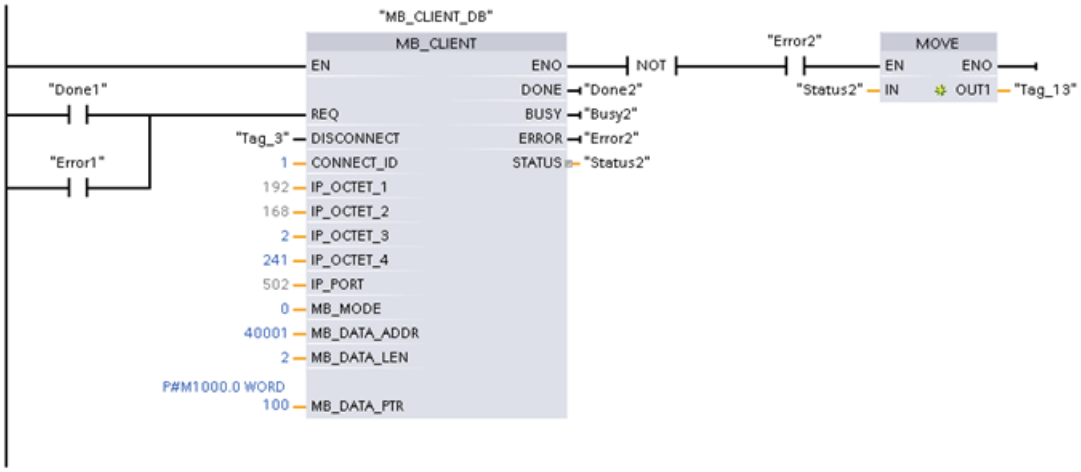
You must ensure that each individual Modbus TCP request finishes execution. This coordination must be provided by your program. The example below shows how the outputs of the first and second client requests can be used to coordinate execution.

The example shows both clients writing to the same memory area. Also, a returned error is captured which is optional.

Network 1: Modbus function 3 - Read holding register words



Network 2: Modbus function 3 - Read holding register words



12.5.3 Modbus RTU

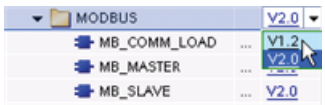
There are two versions of the Modbus RTU instructions available in STEP 7:

- Version 1 was initially available in STEP 7 Basic V10.5.
- Version 2 is available in STEP 7 Basic/Professional V11. The version 2 design adds REQ and DONE parameters to MB_COMM_LOAD. Also, the MB_ADDR parameter for MB_MASTER and MB_SLAVE now allows a UInt value for extended addressing.

For compatibility and ease of migration, you can choose which instruction version to insert into your user program.

Do not use both 1.x and 2.y instruction versions in the same CPU program. Your program's Modbus instructions must have the same major version number (**1**.x, **2**.y, or **V**.z). The individual instructions within a major version group may have different minor versions (1.**x**).

Click the icon on the instruction tree task card to enable the headers and columns of the instruction tree.



To change the version of the Modbus instructions, select the version from the drop-down list. You can select the group or individual instructions.

When you use the instruction tree to place a Modbus instruction in your program, a new FB instance is created in the project tree. You can see new FB instance in the project tree under PLC_x > Program blocks > System blocks > Program resources.

To verify the version of a Modbus instruction in a program, you must inspect project tree properties and not the properties of a box displayed in the program editor. Select a project tree Modbus FB instance, right-click, select "Properties", and select the "Information" page to see the Modbus instruction version number.

12.5.3.1 MB_COMM_LOAD

Table 12- 63 MB_COMM_LOAD instruction

LAD / FBD	SCL	Description

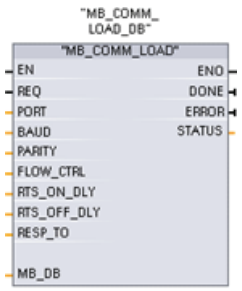
	`"MB_COMM_LOAD_DB"(`
	`REQ:=_bool_in,`
	`PORT:=_uint_in_,`
	`BAUD:=_udint_in_,`
	`PARITY:=_uint_in_,`
	`FLOW_CTRL:=_uint_in_,`
	`RTS_ON_DLY:=_uint_in_,`
	`RTS_OFF_DLY:=_uint_in_,`
	`RESP_TO:=_uint_in_,`
	`DONE=>_bool_out,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`MB_DB:=_fbtref_inout_);`
		The MB_COMM_LOAD instruction configures a PtP port for Modbus RTU protocol communications. Modbus port hardware options: Install up to three CMs (RS485 or RS232), plus one CB (R4845). An instance data block is assigned automatically when you place the MB_COMM_LOAD instruction in your program.

Table 12- 64 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	A low to high (positive edge) signal starts the operation. (Version 2.0 only)
PORT	IN	Port	After you install and configure a CM or CB communication device, the port identifier appears in the parameter helper drop-list available at the PORT box connection. The assigned CM or CB port value is the device configuration property "hardware identifier". The port symbolic name is assigned in the "System constants" tab of the PLC tag table.
BAUD	IN	UDInt	Baud rate selection: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200, all other values are invalid
PARITY	IN	UInt	Parity selection: • 0 – None • 1 – Odd • 2 – Even
FLOW_CTRL	IN	UInt	Flow control selection: • 0 – (default) no flow control • 1 – Hardware flow control with RTS always ON (does not apply to RS485 ports) • 2 – Hardware flow control with RTS switched

Parameter and type		Data type	Description
RTS_ON_DLY	IN	UInt	RTS ON delay selection: • 0 – (default) No delay from RTS active until the first character of the message is transmitted • 1 to 65535 – Delay in milliseconds from RTS active until the first character of the message is transmitted (does not apply to RS485 ports). RTS delays shall be applied independent of the FLOW_CTRL selection.
RTS_OFF_DLY	IN	UInt	RTS OFF delay selection: • 0 – (default) No delay from the last character transmitted until RTS goes inactive • 1 to 65535 – Delay in milliseconds from the last character transmitted until RTS goes inactive (does not apply to RS485 ports). RTS delays shall be applied independent of the FLOW_CTRL selection.
RESP_TO	IN	UInt	Response timeout: Time in milliseconds allowed by MB_MASTER for the slave to respond. If the slave does not respond in this time period, MB_MASTER will retry the request or terminate the request with an error when the specified number of retries has been sent. 5 ms to 65535 ms (default value = 1000 ms).
MB_DB	IN	Variant	A reference to the instance data block used by the MB_MASTER or MB_SLAVE instructions. After MB_SLAVE or MB_MASTER is placed in your program, the DB identifier appears in the parameter helper drop-list available at the MB_DB box connection.
DONE	OUT	Bool	The DONE bit is TRUE for one scan, after the last request was completed with no error. (Version 2.0 only)
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code

MB_COMM_LOAD is executed to configure a port for the Modbus RTU protocol. Once a port is configured for the Modbus RTU protocol, it can only be used by either the MB_MASTER or MB_SLAVE instructions.

One execution of MB_COMM_LOAD must be used to configure each communication port that is used for Modbus communication. Assign a unique MB_COMM_LOAD instance DB for each port that you use. You can install up to three communication modules (RS232 or RS485) and one communication board (RS485) in the CPU. Call MB_COMM_LOAD from a startup OB and execute it one time or use the first scan system flag (Page 78) to initiate the call to execute it one time. Only execute MB_COMM_LOAD again if communication parameters like baud rate or parity must change.

An instance data block is assigned for MB_MASTER or MB_SLAVE when you place these instructions in your program. This instance data block is referenced when you specify the MB_DB parameter for the MB_COMM_LOAD instruction.

MB_COMM_LOAD data block variables

The following table shows the public static variables stored in the instance DB for the MB_COMM_LOAD that can be used in your program.

Table 12- 65 Static variables in the instance DB

Variable	Data type	Description
ICHAR_GAP	Word	Delay for Inter-character gap between characters. This parameter is specified in milliseconds and is used to increase the expected amount of time between received characters. The corresponding number of bit times for this parameter is added to the Modbus default of 35 bit times (3.5 character times).
RETRIES	Word	Number of retries that the master will attempt before returning the no response error code 0x80C8.

Table 12- 66 MB_COMM_LOAD execution condition codes [1]

STATUS (W#16#)	Description
0000	No error
8180	Invalid port ID value (wrong port/hardware identifier for communication module)
8181	Invalid baud rate value
8182	Invalid parity value
8183	Invalid flow control value
8184	Invalid response timeout value (response timeout less than the 5 ms minimum)
8185	MB_DB parameter is not an instance data block of a MB_MASTER or MB_SLAVE instruction.

[1] In addition to the MB_COMM_LOAD errors listed above, errors can be returned from the underlying PtP communication instructions.

See also

Point-to-Point instructions (Page 532)

12.5.3.2 MB_MASTER

Table 12- 67 MB_MASTER instruction

LAD / FBD	SCL	Description

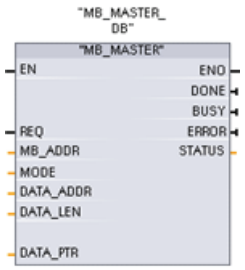
	`"MB_MASTER_DB"(`
	`REQ:=_bool_in_,`
	`MB_ADDR:=_uint_in_,`
	`MODE:=_usint_in_,`
	`DATA_ADDR:=_udint_in_,`
	`DATA_LEN:=_uint_in_,`
	`DONE=>_bool_out_,`
	`BUSY=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`DATA_PTR:=_variant_inout_);`
		The MB_MASTER instruction communicates as a Modbus master using a port that was configured by a previous execution of the MB_COMM_LOAD instruction. An instance data block is assigned automatically when you place the MB_MASTER instruction in your program. This MB_MASTER instance data block is used when you specify the MB_DB parameter for the MB_COMM_LOAD instruction.

Table 12- 68 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	0=No request 1= Request to transmit data to Modbus slave
MB_ADDR	IN	V1.0: USInt V2.0: UInt	Modbus RTU station address: Standard addressing range (1 to 247) Extended addressing range (1 to 65535) The value of 0 is reserved for broadcasting a message to all Modbus slaves. Modbus function codes 05, 06, 15 and 16 are the only function codes supported for broadcast.
MODE	IN	USInt	Mode Selection: Specifies the type of request (read, write, or diagnostic). See the Modbus functions table below for details.
DATA_ADDR	IN	UDInt	Starting Address in the slave: Specifies the starting address of the data to be accessed in the Modbus slave. See the Modbus functions table below for valid addresses.
DATA_LEN	IN	UInt	Data Length: Specifies the number of bits or words to be accessed in this request. See the Modbus functions table below for valid lengths.
DATA_PTR	IN	Variant	Data Pointer: Points to the M or DB address (Standard DB type) for the data being written or read.
DONE	OUT	Bool	The DONE bit is TRUE for one scan, after the last request was completed with no error.
BUSY	OUT	Bool	• 0 – No MB_MASTER operation in progress • 1 – MB_MASTER operation in progress
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution condition code

Modbus master communication rules

- MB_COMM_LOAD must be executed to configure a port before a MB_MASTER instruction can communicate with that port.
- If a port is to be used to initiate Modbus master requests, that port should not be used by MB_SLAVE . One or more instances of MB_MASTER execution can be used with that port, but all MB_MASTER execution must use the same MB_MASTER instance DB for that port.
- The Modbus instructions do not use communication interrupt events to control the communication process. Your program must poll the MB_MASTER instruction for transmit and receive complete conditions.
- It is recommended that you call all MB_MASTER execution for a given port from a program cycle OB. Modbus master instructions may execute in only one of the program cycle or cyclic/time delay execution levels. They must not execute in both execution priority levels. Pre-emption of a Modbus Master instruction by another Modbus master instruction in a higher priority execution priority level will result in improper operation. Modbus master instructions must not execute in the startup, diagnostic or time error execution priority levels.
- Once a master instruction initiates a transmission, this instance must be continually executed with the EN input enabled until a DONE=1 state or ERROR=1 state is returned. A particular MB_MASTER instance is considered active until one of these two events occurs. While the original instance is active, any call to any other instance with the REQ input enabled will result in an error. If the continuous execution of the original instance stops, the request state remains active for a period of time specified by the static variable Blocked_Proc_Timeout. Once this period of time expires, the next master instruction called with an enabled REQ input will become the active instance. This prevents a single Modbus master instance from monopolizing or locking access to a port. If the original active instance is not enabled within the period of time specified by the static variable "Blocked_Proc_Timeout", then the next execution by this instance (with REQ not set) will clear the active state. If (REQ is set), then this execution initiates a new master request as if no other instance was active.

REQ parameter

0 = No request; 1 = Request to transmit data to Modbus Slave

You may control this input either through the use of a level or edge triggered contact. Whenever this input is enabled, a state machine is started to ensure that no other MB_MASTER using the same instance DB is allowed to issue a request, until the current request is completed. All other input states are captured and held internally for the current request, until the response is received or an error detected.

If the same instance of MB_MASTER is executed again with REQ input = 1 before the completion of the current request, then no subsequent transmissions are made. However, when the request is completed, a new request is issued whenever MB_MASTER is executed again with REQ input = 1.

DATA_ADDR and MODE parameters select the Modbus function type

DATA_ADDR (starting Modbus address in the slave): Specifies the starting address of the data to be accessed in the Modbus slave.

The MB_MASTER instruction uses a MODE input rather than a Function Code input. The combination of MODE and Modbus address determine the Function Code that is used in the actual Modbus message. The following table shows the correspondence between parameter MODE, Modbus function code, and Modbus address range.

Table 12- 69 Modbus functions

MODE	Modbus Function	Data length	Operation and data	Modbus Address
0	01	1 to 2000 1 to 1992 [1]	Read output bits: 1 to (1992 or 2000) bits per request	1 to 9999
0	02	1 to 2000 1 to 1992 [1]	Read input bits: 1 to (1992 or 2000) bits per request	10001 to 19999
0	03	1 to 125 1 to 124 [1]	Read Holding registers: 1 to (124 or 125) words per request	40001 to 49999 or 400001 to 465535
0	04	1 to 125 1 to 124 [1]	Read input words: 1 to (124 or 125) words per request	30001 to 39999
1	05	1	Write one output bit: One bit per request	1 to 9999
1	06	1	Write one holding register: 1 word per request	40001 to 49999 or 400001 to 465535
1	15	2 to 1968 2 to 1960 [1]	Write multiple output bits: 2 to (1960 or 1968) bits per request	1 to 9999
1	16	2 to 123 2 to 122 [1]	Write multiple holding registers: 2 to (122 or 123) words per request	40001 to 49999 or 400001 to 465535
2	15	1 to 1968 2 to 1960 [1]	Write one or more output bits: 1 to (1960 or 1968) bits per request	1 to 9999
2	16	1 to 123 1 to 122 [1]	Write one or more holding registers: 1 to (122 or 123) words per request	40001 to 49999 or 400001 to 465535
11	11	0	Read the slave communication status word and event counter. The status word indicates busy (0 – not busy, 0xFFFF - busy). The event counter is incremented for each successful completion of a message. Both the DATA_ADDR and DATA_LEN operands of MB_MASTER are ignored for this function.	
80	08	1	Check slave status using data diagnostic code 0x0000 (Loopback test – slave echoes the request) 1 word per request	
81	08	1	Reset slave event counter using data diagnostic code 0x000A 1 word per request	
3 to 10, 12 to 79, 82 to 255			Reserved	

[1] For "Extended Addressing" mode the maximum data lengths are reduced by 1 byte or 1 word depending upon the data type used by the function.

DATA_PTR parameter

The DATA_PTR parameter points to the DB or M address that is written to or read from. If you use a data block, then you must create a global data block that provides data storage for reads and writes to Modbus slaves.

Note

The DATA_PTR data block type must allow direct addressing

The data block must allow both direct (absolute) and symbolic addressing. When you create the data block the "Standard" access attribute must be selected.

Data block structures for the DATA_PTR parameter

- These data types are valid for **word reads** of Modbus addresses 30001 to 39999, 40001 to 49999, and 400001 to 465536 and also for **word writes** to Modbus addresses 40001 to 49999 and 400001 to 465536.
  - Standard array of WORD, UINT, or INT data types
  - Named WORD, UINT, or INT structure where each element has a unique name and 16 bit data type.
  - Named complex structure where each element has a unique name and a 16 or 32 bit data type.
- For **bit reads** and writes of Modbus addresses 00001 to 09999 and bit reads of 10001 to 19999.
  - Standard array of Boolean data types.
  - Named Boolean structure of uniquely named Boolean variables..
- Although not required, it is recommended that each MB_MASTER instruction have its own separate memory area. The reason for this recommendation is that there is a greater possibility of data corruption if multiple MB_MASTER instructions are reading and writing to the same memory area.
- There is no requirement that the DATA_PTR data areas be in the same global data block. You can create one data block with multiple areas for Modbus reads, one data block for Modbus writes, or one data block for each slave station.

Modbus master data block variables

The following table shows the public static variables stored in the instance DB for MB_MASTER that can be used in your program.

Table 12- 70 Static variables in the instance DB

Variable	Data type	Initial value	Description
Blocked_Proc_Timeout	Real	3.0	Amount of time (in seconds) to wait for a blocked Modbus Master instance before removing this instance as being ACTIVE. This can occur, for example, when a Master request has been issued and then the program stops calling the Master function before it has completely finished the request. The time value must be greater than 0 and less than 55 seconds, or an error occurs. The default value is .5 seconds.
Extended_Addressing	Bool	False	Configures single or double-byte slave addressing. The default value = 0. (0=single byte address, 1=double-byte address)

Your program can write values to the Blocked_Proc_Timeout and Extended_Addressing variables to control Modbus master operations. See the MB_SLAVE topic description of HR_Start_Offset and Extended_Addressing for an example of how to use these variables in the program editor and details about Modbus extended addressing (Page 608).

Condition codes

Table 12- 71 MB_MASTER execution condition codes (communication and configuration errors) [1]

STATUS (W#16#)	Description
0000	No error
80C8	Slave timeout. Check baud rate, parity, and wiring of slave.
80D1	The receiver issued a flow control request to suspend an active transmission and never re-enabled the transmission during the specified wait time. This error is also generated during hardware flow control when the receiver does not assert CTS within the specified wait time.
80D2	The transmit request was aborted because no DSR signal is received from the DCE.
80E0	The message was terminated because the receive buffer is full.
80E1	The message was terminated as a result of a parity error.
80E2	The message was terminated as a result of a framing error.
80E3	The message was terminated as a result of an overrun error.
80E4	The message was terminated as a result of the specified length exceeding the total buffer size.
8180	Invalid port ID value or error with MB_COMM_LOAD instruction
8186	Invalid Modbus station address
8188	Invalid Mode specified for broadcast request
8189	Invalid Data Address value
818A	Invalid Data Length value
818B	Invalid pointer to the local data source/destination: Size not correct
818C	Invalid pointer for DATA_PTR or invalid Blocked_Proc_Timeout: The data area must be a DB (that allows both symbolic and direct access) or a M memory.
8200	Port is busy processing a transmit request.

Table 12- 72 MB_MASTER execution condition codes (Modbus protocol errors) [1]

STATUS (W#16#)	Response code from slave	Modbus protocol errors
8380	-	CRC error
8381	01	Function code not supported
8382	03	Data length error
8383	02	Data address error or address outside the valid range of the DATA_PTR area
8384	Greater than 03	Data value error
8385	03	Data diagnostic code value not supported (function code 08)
8386	-	Function code in the response does not match the code in the request.
8387	-	Wrong slave responded
8388	-	The slave response to a write request is incorrect. The write request returned by the slave does not match what the master actually sent.

[1] In addition to the MB_MASTER errors listed above, errors can be returned from the underlying PtP communication instructions.

See also

Point-to-Point instructions (Page 532)

12.5.3.3 MB_SLAVE

Table 12- 73 MB_SLAVE instruction

LAD / FBD	SCL	Description

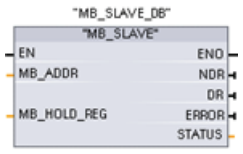
	`"MB_SLAVE_DB"(`
	`MB_ADDR:=_uint_in_,`
	`NDR=>_bool_out_,`
	`DR=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,`
	`MB_HOLD_REG:=_variant_inout_);`
		The MB_SLAVE instruction allows your program to communicate as a Modbus slave through a PtP port on the CM (RS485 or RS232) and CB (RS485). When a remote Modbus RTU master issues a request, your user program responds to the request by MB_SLAVE execution. STEP 7 automatically creates an instance DB when you insert the instruction. Use this MB_SLAVE_DB name when you specify the MB_DB parameter for the MB_COMM_LOAD instruction.

Table 12- 74 Data types for the parameters

Parameter and type		Data type	Description
MB_ADDR	IN	V1.0: USInt V2.0: UInt	The station address of the Modbus slave: Standard addressing range (1 to 247) Extended addressing range (0 to 65535)
MB_HOLD_REG	IN	Variant	Pointer to the Modbus Holding Register DB: The Modbus holding register can be M memory or a data block.
NDR	OUT	Bool	New Data Ready: • 0 – No new data • 1 – Indicates that new data has been written by the Modbus master
DR	OUT	Bool	Data Read: • 0 – No data read • 1 – Indicates that data has been read by the Modbus master
ERROR	OUT	Bool	The ERROR bit is TRUE for one scan, after the last request was terminated with an error. If execution is terminated with an error, then the error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Execution error code

Modbus communication function codes (1, 2, 4, 5, and 15) can read and write bits and words directly in the input process image and output process image of the CPU. For these function codes, the MB_HOLD_REG parameter must be defined as a data type larger than a byte. The following table shows the example mapping of Modbus addresses to the process image in the CPU.

Table 12- 75 Mapping of Modbus addresses to the process image

Modbus functions						S7-1200	
Codes	Function	Data area	Address range			Data area	CPU address
01	Read bits	Output	1	to	8192	Output Process Image	Q0.0 to Q1023.7
02	Read bits	Input	10001	to	18192	Input Process Image	I0.0 to I1023.7
04	Read words	Input	30001	to	30512	Input Process Image	IW0 to IW1022
05	Write bit	Output	1	to	8192	Output Process Image	Q0.0 to Q1023.7
15	Write bits	Output	1	to	8192	Output Process Image	Q0.0 to Q1023.7

Modbus communication function codes (3, 6, 16) use a Modbus holding register which can be a M memory address range or a data block. The type of holding register is specified by the MB_HOLD_REG parameter on the MB_SLAVE instruction.

Note

MB_HOLD_REG data block type

A Modbus holding register data block must allow both direct (absolute) and symbolic addressing. When you create the data block the "Standard" access attribute must be selected.

The following table shows examples of Modbus address to holding register mapping that is used for Modbus function codes 03 (read words), 06 (write word), and 16 (write words). The actual upper limit of DB addresses is determined by the maximum work memory limit and M memory limit, for each CPU model.

Table 12- 76 Mapping of Modbus addresses to CPU memory

Modbus Master Address	MB_HOLD_REG parameter examples				
	MW100	DB10.DBw0	MW120	DB10.DBW50	"Recipe".ingredient
40001	MW100	DB10.DBW0	MW120	DB10.DBW50	"Recipe".ingredient[1]
40002	MW102	DB10.DBW2	MW122	DB10.DBW52	"Recipe".ingredient[2]
40003	MW104	DB10.DBW4	MW124	DB10.DBW54	"Recipe".ingredient[3]
40004	MW106	DB10.DBW6	MW126	DB10.DBW56	"Recipe".ingredient[4]
40005	MW108	DB10.DBW8	MW128	DB10.DBW58	"Recipe".ingredient[5]

Table 12- 77 Diagnostic functions

S7-1200 MB_SLAVE Modbus diagnostic functions		
Codes	Sub-function	Description
08	0000H	Return query data echo test: The MB_SLAVE will echo back to a Modbus master a word of data that is received.
08	000AH	Clear communication event counter: The MB_SLAVE will clear out the communication event counter that is used for Modbus function 11.
11		Get communication event counter: The MB_SLAVE uses an internal communication event counter for recording the number of successful Modbus read and write requests that are sent to the Modbus slave. The counter does not increment on any Function 8, Function 11, or broadcast requests. It is also not incremented on any requests that result in a communication error (for example, parity or CRC errors).

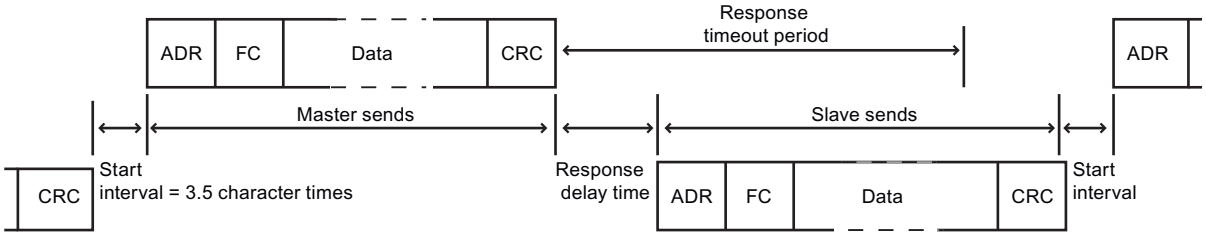
The MB_SLAVE instruction supports broadcast write requests from any Modbus master as long as the request is for accessing valid addresses. MB_SLAVE will produce error code 0x8188 for function codes not supported in broadcast.

Modbus slave communication rules

- MB_COMM_LOAD must be executed to configure a port, before a MB_SLAVE instruction can communicate through that port.
- If a port is to respond as a slave to a Modbus master, then do not program that port with the MB_MASTER instruction.
- Only one instance of MB_SLAVE can be used with a given port, otherwise erratic behavior may occur.
- The Modbus instructions do not use communication interrupt events to control the communication process. Your program must control the communication process by polling the MB_SLAVE instruction for transmit and receive complete conditions.
- The MB_SLAVE instruction must execute periodically at a rate that allows it to make a timely response to incoming requests from a Modbus master. It is recommended that you execute MB_SLAVE every scan from a program cycle OB. Executing MB_SLAVE from a cyclic interrupt OB is possible, but is not recommended because of the potential for excessive time delays in the interrupt routine to temporarily block the execution of other interrupt routines.

Modbus signal timing

MB_SLAVE must be executed periodically to receive each request from the Modbus master and then respond as required. The frequency of execution for MB_SLAVE is dependent upon the response timeout period of the Modbus master. This is illustrated in the following diagram.



The response timeout period RESP_TO is the amount of time a Modbus master waits for the start of a response from a Modbus slave. This time period is not defined by the Modbus protocol, but is a parameter of each Modbus master. The frequency of execution (the time between one execution and the next execution) of MB_SLAVE must be based on the particular parameters of your Modbus master. At a minimum, you should execute MB_SLAVE twice within the response timeout period of the Modbus master.

Modbus slave variables

This table shows the public static variables stored in the MB_SLAVE instance data block that can be used in your program

Table 12- 78 Modbus slave variables

Variable	Data type	Description
HR_Start_Offset	Word	Specifies the starting address of the Modbus Holding register (default = 0)
Extended_Addressing	Bool	Configures single or double-byte slave addressing (0=single byte address, 1=double-byte address, default = 0)
Request_Count	Word	The number of all requests received by this slave
Slave_Message_Count	Word	The number of requests received for this specific slave
Bad_CRC_Count	Word	The number of requests received that have a CRC error
Broadcast_Count	Word	The number of broadcast requests received
Exception_Count	Word	Modbus specific errors that require a returned exception
Success_Count	Word	The number of requests received for this specific slave that have no protocol errors

Your program can write values to the HR_Start_Offset and Extended_Addressing variables and control Modbus slave operations. The other variables can be read to monitor Modbus status.

HR_Start_Offset

Modbus holding register addresses begin at 40001 or 400001. These addresses correspond to the beginning PLC memory address of the holding register. However, you can configure the "HR_Start_Offset" variable to start the beginning Modbus holding register address at another value instead of 40001 or 400001.

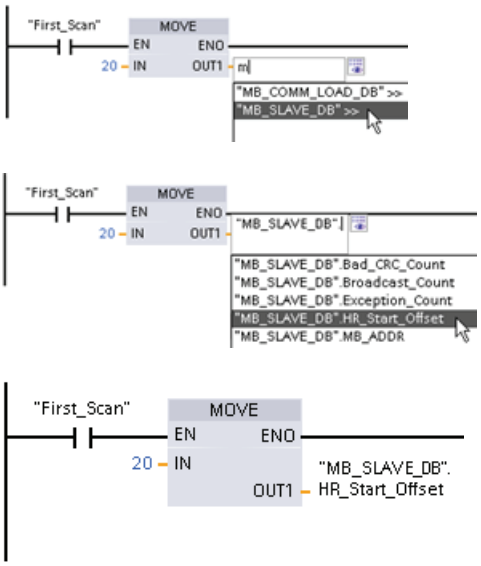
For example, if the holding register is configured to start at MW100 and is 100 words long. An offset of 20 specifies a beginning holding register address of 40021 instead of 40001. Any address below 40021 and above 400119 will result in an addressing error.

Table 12- 79 Example of Modbus holding register addressing

HR_Start_Offset	Address	Minimum	Maximum
0	Modbus address (Word)	40001	40099
	S7-1200 address	MW100	MW298
20	Modbus address (Word)	40021	40119
	S7-1200 address	MW100	MW298

HR_Start_Offset is a word value that specifies the starting address of the Modbus holding register and is stored in the MB_SLAVE instance data block. You can set this public static variable value by using the parameter helper drop-list, after MB_SLAVE is placed in your program.

For example, after MB_SLAVE is placed in a LAD network, you can go to a previous network and assign the HR_Start_Offset value. The value must be assigned prior to execution of MB_SLAVE.



Entering a Modbus slave variable using the default DB name:

1. Set the cursor in the parameter field and type an m character.
2. Select "MB_SLAVE_DB" from the drop-list.
3. Set the cursor at the right side of the DB name (after the quote character) and enter a period character.
4. Select "MB_SLAVE_DB.HR_Start_Offset" from the drop list.

Extended_Addressing

The Extended_Addressing variable is accessed in a similar way as the HR_Start_Offset reference discussed above except that the Extended_Addressing variable is a boolean value. The boolean value must be written by an output coil and not a move box.

Modbus slave addressing can be configured to be either a single byte (which is the Modbus standard) or double byte. Extended addressing is used to address more than 247 devices within a single network. Selecting extended addressing allows you to address a maximum of 64000 addresses. A Modbus function 1 frame is shown below as an example.

Table 12- 80 Single-byte slave address (byte 0)

Function 1	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	
Request	Slave addr.	F code	Start address		Length of coils		
Valid Response	Slave addr.	F code	Length	Coil data			
Error response	Slave addr.	0x81	E code				

Table 12- 81 Double-byte slave address (byte 0 and byte 1)

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Request	Slave address		F code	Start address		Length of coils	
Valid Response	Slave address		F code	Length	Coil data		
Error response	Slave address		0x81	E code			

Condition codes

Table 12- 82 MB_SLAVE execution condition codes (communication and configuration errors) [1]

STATUS (W#16#)	Description
80D1	The receiver issued a flow control request to suspend an active transmission and never re-enabled the transmission during the specified wait time. This error is also generated during hardware flow control when the receiver does not assert CTS within the specified wait time.
80D2	The transmit request was aborted because no DSR signal is received from the DCE.
80E0	The message was terminated because the receive buffer is full.
80E1	The message was terminated as a result of a parity error.
80E2	The message was terminated as a result of a framing error.
80E3	The message was terminated as a result of an overrun error.
80E4	The message was terminated as a result of the specified length exceeding the total buffer size.
8180	Invalid port ID value or error with MB_COMM_LOAD instruction
8186	Invalid Modbus station address
8187	Invalid pointer to MB_HOLD_REG DB: Area is too small
818C	Invalid MB_HOLD_REG pointer to M memory or DB (DB area must allow both symbolic and direct address)

Table 12- 83 MB_SLAVE execution condition codes (Modbus protocol errors) [1]

STATUS (W#16#)	Response code from slave	Modbus protocol errors
8380	No response	CRC error
8381	01	Function code not supported or not supported within broadcasts
8382	03	Data length error
8383	02	Data address error or address outside the valid range of the DATA_PTR area
8384	03	Data value error
8385	03	Data diagnostic code value not supported (function code 08)

[1] In addition to the MB_SLAVE errors listed above, errors can be returned from the underlying PtP communication instructions.

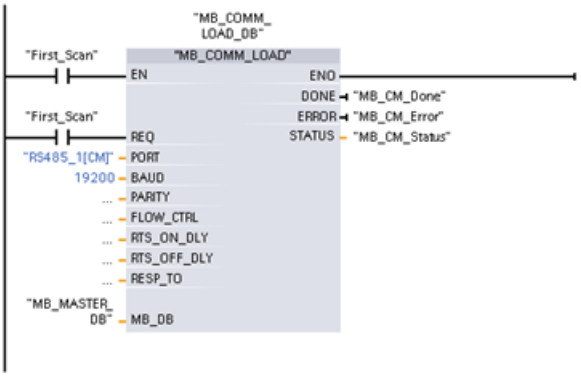
See also

Point-to-Point instructions (Page 532)

12.5.3.4 Modbus RTU master example program

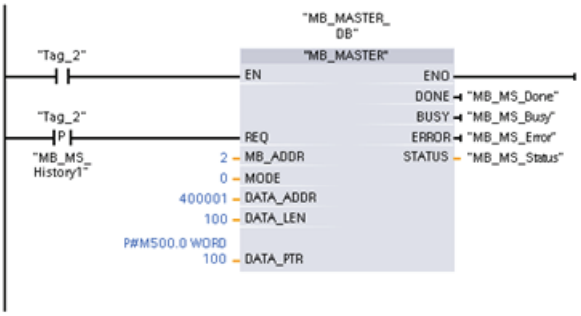
MB_COMM_LOAD is initialized during start-up by using the first scan flag. Execution of MB_COMM_LOAD in this manner should only be done when the serial port configuration will not change at runtime.

Network 1 Initialize the RS-485 module parameters only once during the first scan.

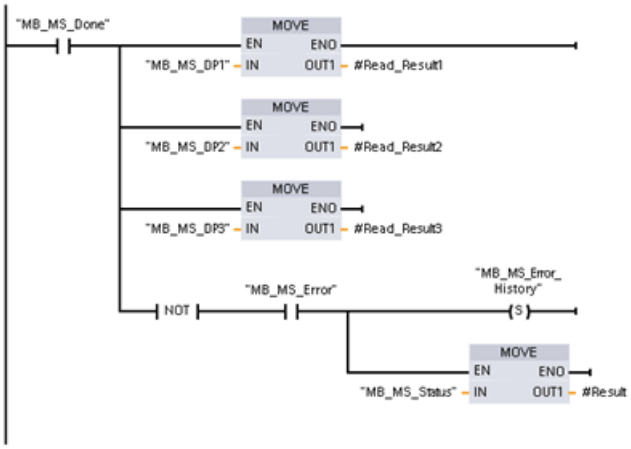


One MB_MASTER instruction is used in the program cycle OB to communicate with a single slave. Additional MB_MASTER instructions can be used in the program cycle OB to communicate with other slaves, or one MB_MASTER FB could be re-used to communicate with additional slaves.

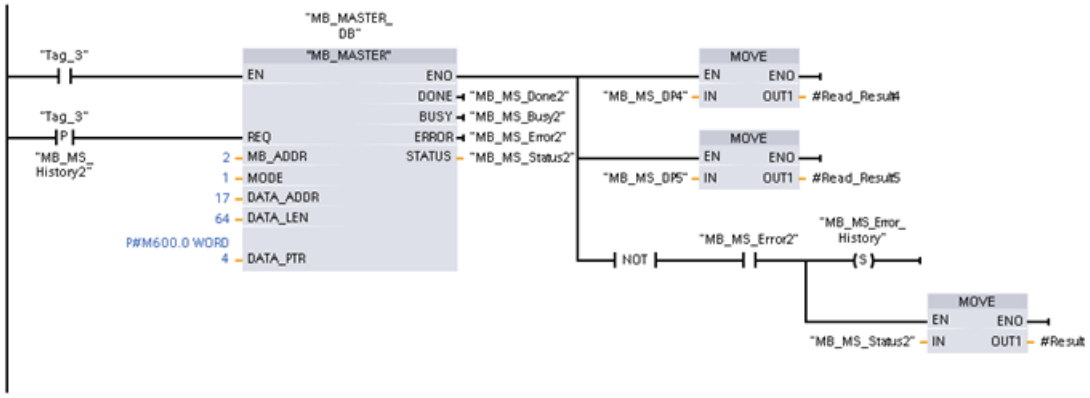
Network 2 Read 100 words from the slave holding register.



Network 3 This is an optional network that just shows the values of the first 3 words once the read operation is done.



Network 4 Write 64 bits to the output image register starting at slave address Q2.0.

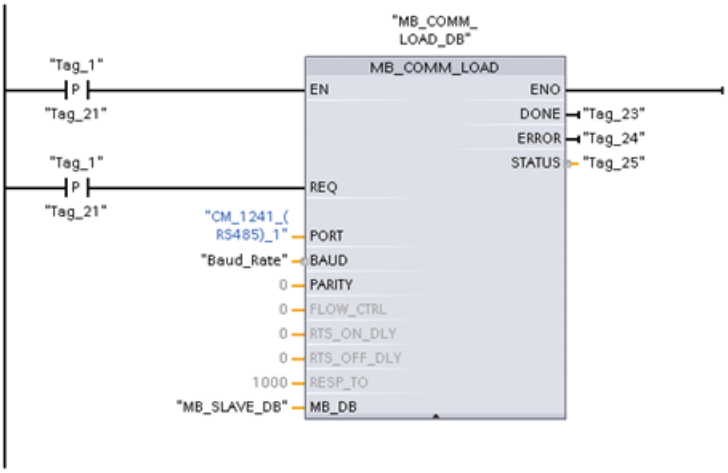


12.5.3.5 Modbus RTU slave example program

MB_COMM_LOAD shown below is initialized each time "Tag_1" is enabled.

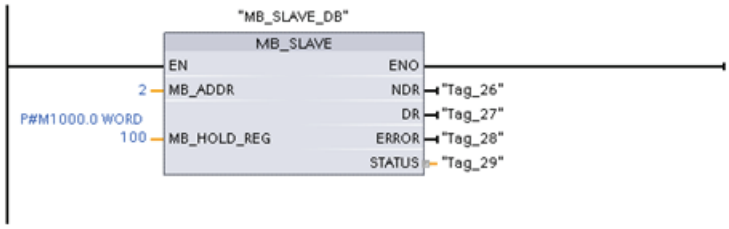
Execution of MB_COMM_LOAD in this manner should only be done when the serial port configuration will change at runtime, as a result of HMI configuration.

Network 1 Initialize the RS-485 module parameters each time they are changed by an HMI device.



MB_SLAVE shown below is placed in a cyclic OB that is executed every 10ms. While this does not give the absolute fastest response by the slave, it does provide good performance at 9600 baud for short messages (20 bytes or less in the request).

Network 2 Check for Modbus master requests during each scan. The Modbus holding register is configured for 100 words starting at MW1000.



12.6 Telecontrol and TeleService with the CP 1242-7

12.6.1 Connection to a GSM network

IP-based WAN communication via GPRS

Using the CP 1242-7 communications processor, the S7-1200 can be connected to GSM networks. The CP 1242-7 allows WAN communication from remote stations with a control center and inter-station communication.

Inter-station communication is possible only via a GSM network. For communication between a remote station and a control room, the control center must have a PC with Internet access.

The CP 1242-7 supports the following services for communication via the GSM network:

- GPRS (General Packet Radio Service)

  The packet-oriented service for data transmission "GPRS" is handled via the GSM network.

- SMS (Short Message Service)

  The CP 1242-7 can receive and send SMS messages. The communications partner can be a mobile phone or an S7-1200.

The CP 1242-7 is suitable for use in industry worldwide and supports the following frequency bands:

- 850 MHz
- 900 MHz
- 1 800 MHz
- 1 900 MHz

Requirements

The equipment used in the stations or the control center depends on the particular application.

- For communication with or via a central control room, the control center requires a PC with Internet access.
- Apart from the station equipment, a remote S7-1200 station with a CP 1242-7 must meet the following requirements to be able to communicate via the GSM network:
  - A contract with a suitable GSM network provider

    If GPRS is used, the contract must allow the use of the GPRS service.

    If there is to be direct communication between stations only via the GSM network, the GSM network provider must assign a fixed IP address to the CPs. In this case, communication between stations is not via the control center.
  - The SIM card belonging to the contract

    The SIM card is inserted in the CP 1242-7.
  - Local availability of a GSM network in the range of the station

12.6.2 Applications of the CP 1242-7

The CP 1242-7 can be used for the following applications:

Telecontrol applications

- Sending messages by SMS

  Via the CP 1242-7, the CPU of a remote S7-1200 station can receive SMS messages from the GSM network or send messages by SMS to a configured mobile phone or an S7-1200.

- Communication with a control center

  Remote S7-1200 stations communicate via the GSM network and the Internet with a telecontrol server in the master station. For data transfer using GPRS, the "TELECONTROL SERVER BASIC" application is installed on the telecontrol server in the master station. The telecontrol server communicates with a higher-level central control system using the integrated OPC server function.

- Inter-station communication between S7-1200 stations via a GSM network

  Inter-station communication between remote stations with a CP 1242-7 can be handled in two different ways:

  – Indirect communication via a master station

    In this configuration, a permanent secure connection between S7-1200 stations that communicate with each other and the telecontrol server is established in the master station. Communication between the stations is via the telecontrol server. The CP 1242-7 operates in "Telecontrol" mode.

  – Direct communication between the stations

    For direct communication between stations without the detour via the master station, SIM cards with a fixed IP address are used that allow the stations to address each other directly. The possible communications services and security functions (for example VPN) depend on what is offered by the network provider. The CP 1242-7 operates in "GPRS direct" mode.

TeleService via GPRS

A TeleService connection can be established between an engineering station with STEP 7 and a remote S7-1200 station with a CP 1242-7 via the GSM network and the Internet. The connection runs from the engineering station via a telecontrol server or a TeleService gateway that acts as an intermediary forwarding frames and establishing the authorization. These PCs use the functions of the "TELECONTROL SERVER BASIC" application.

You can use the TeleService connection for the following purposes:

- Downloading configuration or program data from the STEP 7 project to the station
- Querying diagnostics data on the station

12.6.3 Other properties of the CP

Other services and functions of the CP 1242-7

- Time-of-day synchronization of the CP via the Internet

  You can set the time on the CP as follows:

  – In "Telecontrol" mode, the time of day is transferred by the telecontrol server. The CP uses this to set its time.

  – In "GPRS direct" mode, the CP can request the time using SNTP.

  To synchronize the CPU time, you can read out the current time from the CP using a block.
- Interim buffering of messages to be sent if there are connection problems
- Increased availability thanks to the option of connecting to a substitute telecontrol server
- Optimized data volume (temporary connection)

  As an alternative to a permanent connection to the telecontrol server, the CP can be configured in STEP 7 with a temporary connection to the telecontrol server. In this case, a connection to the telecontrol server is established only when required.
- Logging the volume of data

  The volumes of data transferred are logged and can be evaluated for specific purposes.

Configuration and module replacement

To configure the module, the following configuration tool is required:

STEP 7 version V11.0 SP1 or higher

For STEP 7 V11.0 SP1, you also require support package "CP 1242-7" (HSP0003001).

For process data transfer using GPRS, use the telecontrol communications instructions in the user program of the station.

The configuration data of the CP 1242-7 is stored on the local CPU. This allows simple replacement of the CP when necessary.

You can insert up to three modules of the CP 1242-7 type per S7-1200. This, for example, allows redundant communications paths to be established.

Electrical connections

- Power supply of the CP 1242-7

  The CP has a separate connection for the external 24 VDC power supply.
- Wireless interface for the GSM network

  An extra antenna is required for GSM communication. This is connected via the SMA socket of the CP.

Further information

The CP 1242-7 manual contains detailed information. You will find this on the Internet on the pages of Siemens Industrial Automation Customer Support under the following entry ID:

42330276 (http://support.automation.siemens.com/WW/view/en/42330276)

12.6.4 Accessories

The ANT794-4MR GSM/GPRS antenna

The following antennas are available for use in GSM/GPRS networks and can be installed both indoors and outdoors:

- Quadband antenna ANT794-4MR

Figure 12-1 ANT794-4MR GSM/GPRS antenna

Short name	Order no.	Explanation
ANT794-4MR	6NH9 860-1AA00	Quadband antenna (900, 1800/1900 MHz, UMTS); weatherproof for indoor and outdoor areas; 5 m connecting cable connected permanently to the antenna; SMA connector, including installation bracket, screws, wall plugs

- Flat antenna ANT794-3M

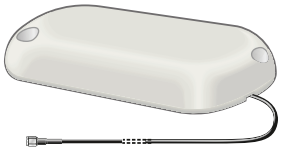


Figure 12-2 Flat antenna ANT794-3M

Short name	Order no.	Explanation
ANT794-3M	6NH9 870-1AA00	Flat antenna (900, 1800/1900 MHz); weatherproof for indoor and outdoor areas; 1.2 m connecting cable connected permanently to the antenna; SMA connector, including adhesive pad, screws mounting possible

The antennas must be ordered separately.

Further information

You will find detailed information in the device manual. You will find this on the Internet on the pages of Siemens Industrial Automation Customer Support under the following entry ID:

23119005 (http://support.automation.siemens.com/WW/view/en/23119005)

12.6.5 Configuration examples for telecontrol

Below, you will find several configuration examples for stations with a CP 1242-7.

Sending messages by SMS

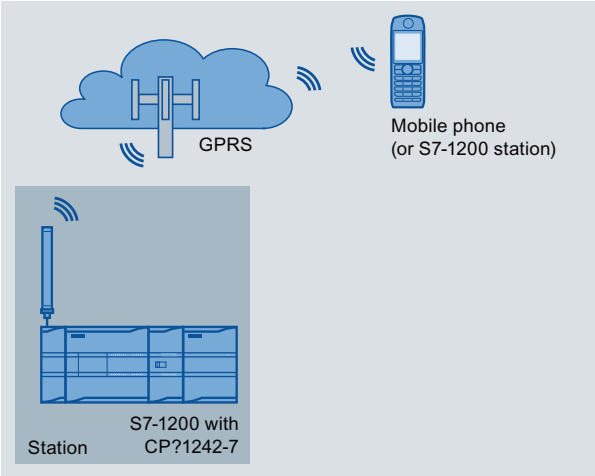


Figure 12-3 Sending messages by SMS from an S7-1200 station

A SIMATIC S7-1200 with a CP 1242-7 can send messages by SMS to a configured mobile phone or a configured S7-1200 station.

Telecontrol by a control center

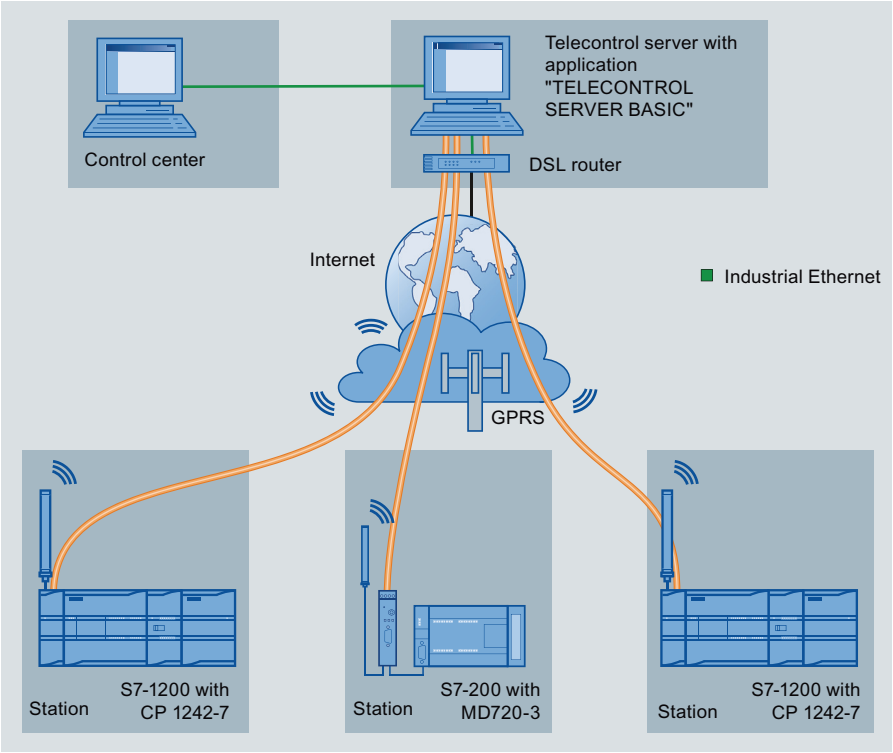


Figure 12-4 Communication between S7-1200 stations and a control center

In telecontrol applications, SIMATIC S7-1200 stations with a CP 1242-7 communicate with a control center via the GSM network and the Internet. The TELECONTROL SERVER BASIC application is installed on the telecontrol server in the master station. This results in the following use cases:

- Telecontrol communication between station and control center

  In this use case, data from the field is sent by the stations to the telecontrol server in the master station via the GSM network and Internet. The telecontrol server is used to control and monitor remote stations.

- Communication between a station and a control center PC with OPC client

  As in the first case, the stations communicate with the telecontrol server. Using the OPC server of TELECONTROL SERVER BASIC, the telecontrol server exchanges data with a master station PC. The control center PC could, for example, have WinCC and an integrated OPC client installed on it.

- Inter-station communication via a control center

  To allow inter-station communication, the telecontrol server forwards the messages of the sending station to the receiving station.

Direct communication between stations

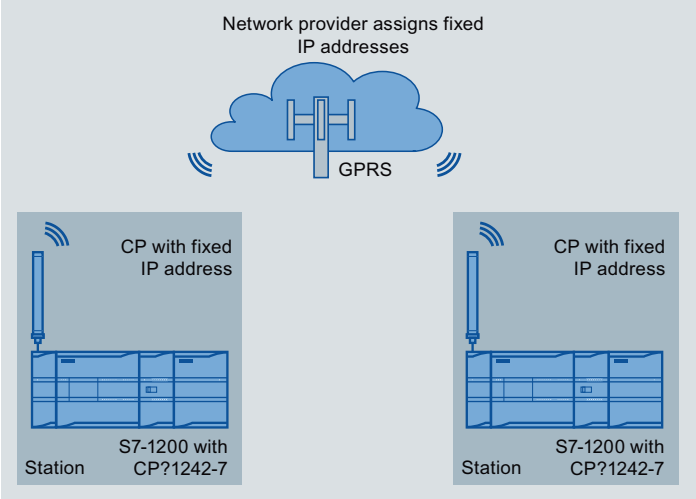


Figure 12-5 Direct communication between two S7-1200 stations

In this configuration, two SIMATIC S7-1200 stations communicate directly with each other using the CP 1242-7 via the GSM network. Each CP 1242-7 has a fixed IP address. The relevant service of the GSM network provider must allow this.

TeleService via GPRS

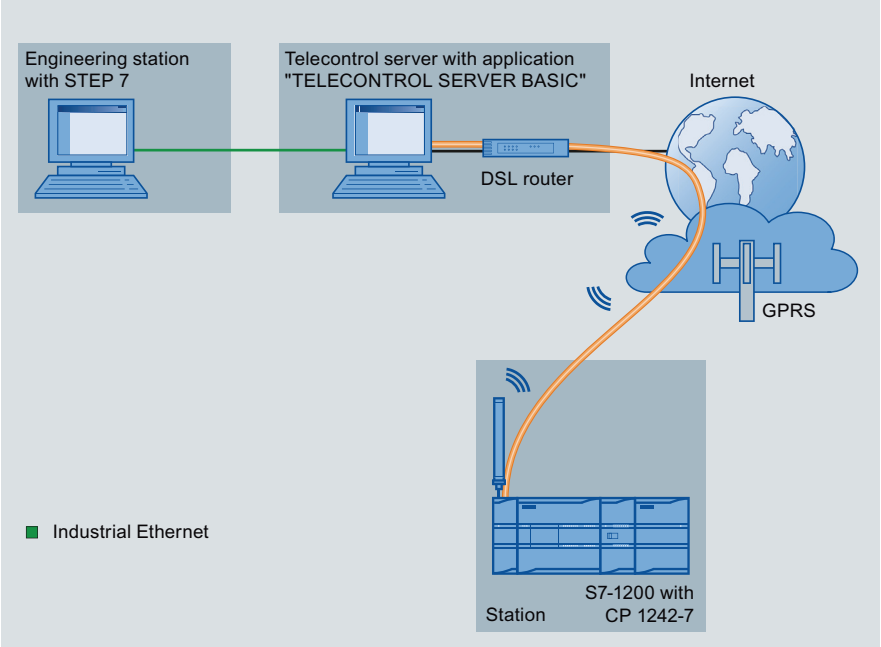


Figure 12-6 TeleService via GPRS

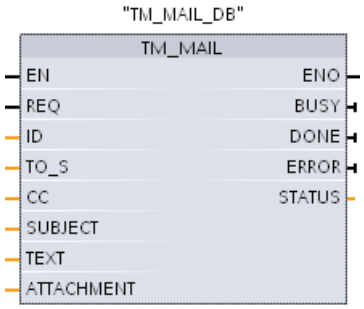
In TeleService via GPRS, an engineering station on which STEP 7 is installed communicates via the GSM network and the Internet with a SIMATIC S7-1200 station with a CP 1242-7. The connection runs via a telecontrol server that serves as an intermediary and is connected to the Internet.

Teleservice communication (SMTP e-mail) 13

13.1 TM_Mail transfer e-mail instruction

Table 13- 1 TM_MAIL instruction

LAD / FBD	SCL	Description

	`"TM_MAIL_DB"(`
	`REQ:=_bool_in_,`
	`ID:=_int_in_,`
	`TO_S:=_string_in_,`
	`CC:=_string_in_,`
	`SUBJECT:=_string_in_,`
	`TEXT:= _string_in_,`
	`ATTACHMENT:=_variant_in_,`
	`BUSY=>_bool_out_,`
	`DONE=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_,);`
		The TM_MAIL instruction sends an e-mail message using the SMTP (Simple Mail Transfer Protocol) via a CPU Industrial Ethernet connection to the optional teleservice adapter. TM_MAIL executes asynchronously and the job extends over multiple TM_MAIL calls.
		When you call TM_MAIL, you must specify an instance DB. **The instance DB retentive attribute must not be set**. This ensures that the instance DB is initialized in the transition of the CPU from STOP to RUN and that a new TM_MAIL operation can be triggered.

[1] STEP 7 automatically creates the instance DB when you insert the instruction.

You start sending an e-mail with a positive edge change from 0 to 1, at input parameter REQ. The following table shows the relationship between BUSY, DONE and ERROR. You can monitor the progress of TM_MAIL execution and detect completion, by evaluating these parameters in successive calls.

The output parameters DONE, ERROR, STATUS, and SFC_STATUS are valid for only one cycle, when the state of the output parameter BUSY changes from 1 to 0. Your program logic must save temporary output state values in data latches, so you can detect state changes in subsequent program execution cycles.

Table 13- 2 Interaction of the Done, Busy and Error parameters

DONE	BUSY	ERROR	Description
Irrelevant	1	Irrelevant	Job is in progress.
1	0	0	The job was completed successfully.
0	0	1	The job was terminated with an error. For the cause of the error, refer to the STATUS parameter.
0	0	0	No job in progress

If the CPU is changed to STOP mode while TM_MAIL is active, then the communication connection to the mail server is terminated. The communication connection to the mail server is also lost if problems occur in CPU communication on the Industrial Ethernet bus. In these cases, the send process is suspended and the e-mail does not reach the recipient.

CAUTION
Modifiying user programs Only change the parts of your user program that directly affect the TM_MAIL calls in the following cases: • The CPU in the STOP mode • No mail is sent (REQ and BUSY = 0) This refers, in particular, to the deletion and replacement of program blocks, the calls to TM_MAIL, or calls to the instance DBs of TM_MAIL. If you fail to maintain linked program blocks, then the TPC / IP communication functions can enter an undefined state. After transferring a modified program block, you must perform a CPU restart (warm) or cold start.

Data consistency

The input parameter ADDR_MAIL_SERVER is read when the operation is started. A new value does not take effect until the current operation is complete and a new TM_MAIL operation is initiated.

In contrast, the parameters WATCH_DOG_TIME, TO_S, CC, FROM, SUBJECT, TEXT, ATTACHMENT, USERNAME and PASSWORD are read during the execution of TM_MAIL and may be changed only when the job is finished (BUSY = 0)

Configuring the TS Adapter IE parameters

You must configure the Teleservice adapter IE parameters for outgoing calls to connect with the dial-up server of your Internet Service Provider. If you set the call "on demand" attribute, then the connection is established only when an e-mail will be sent. For an analog modem connection, more time is required for the connection process (approx. a minute longer). You must include the extra time, in the WATCH_DOG_TIME value.

Table 13- 3 Data types for the parameters

Parameter and type		Data types	Description
REQ	IN	Bool	A low to high (positive edge) signal starts the operation.
ID	IN	Int	Connection identifier: See the ID parameter of the instructions TCON, TDISCON, TSEND and TRCV. A number that is not used for any additional instances of this instruction in the user program must be entered here.
TO_S	IN	String	Recipient addresses: STRING data with a maximum length of 240 characters

Parameter and type		Data types	Description
CC	IN	String	CC copy to recipient addresses (optional): STRING data with a maximum length of 240 characters
SUBJECT	IN	String	Subject name of the e-mail: STRING data with a maximum length 240 characters.
TEXT	IN	String	Text message of the e-mail (optional): STRING data with a maximum length of 240 characters. If this parameter is an empty string, then the e-mail will be sent without message text.
ATTACHMENT	IN	Variant	Pointer to e-mail attachment data: Byte, word, or double word data with a maximum length of 65534 bytes. If no value is assigned, then the e-mail sent without an attachment.
DONE	OUT	Bool	• 0 - Job not yet started or still executing. • 1 - Job was executed error-free.
BUSY	OUT	Bool	• 0 - No operation in progress • 1- Operation in progress
ERROR	OUT	Bool	The ERROR bit =1 for one scan, after the last request was terminated with an error. The error code value at the STATUS output is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	Word	Return value or error information of the TM_MAIL instruction.
ADDR_MAIL_SERVER	Static	DWord	IP address input parameter of the mail server: Specify as a data word in HEX format, for example: IP address = 192.168.0.200. ADDR_MAIL_SERVER = DW#16#C0A800C8, where: • 192 = 16#C0, • 168 =16#A8 • 0 = 16#00 and • 200 = 16#C8
WATCH_DOG_TIME	Static	Time	The maximum time allowed for TM_MAIL to establish a server connection. If this time is exceeded, then TM_MAIL execution ends with an error. The actual time delay until TM_MAIL ends and the error is issued may exceed the WATCH_DOG_TIME, because of the additional time required for the disconnect operation. At first you should set a time of 2 minutes. This time can be much smaller for an ISDN phone connection.
USERNAME	Static	String	Mail account user name: STRING data with a maximum length 180 characters.
PASSWORD	Static	String	Mail server password: STRING data with a maximum length 180 characters.
FROM	Static	String	Sender address: STRING with a maximum length of 240 characters
SFC_STATUS	Static	Word	Execution condition code of the called communication blocks

SMTP authentication

TM_MAIL supports the SMTP AUTH LOGIN authentication method that is required by most mail servers. For information on the authentication method used by your mail server, please refer to the manual of the mail server or the website of your Internet Service Provider.

The AUTH LOGIN authentication method uses the TM_MAIL USERNAME and PASSWORD parameters to access the mail server. The user name and password must be set up on an e-mail account at a mail server.

If no value is assigned for the USERNAME parameter, then the AUTH LOGIN authentication method is not used and the e-mail is sent without authentication.

TO_S, CC, and FROM parameters

The parameters TO_S, CC and FROM are strings, as shown in the following examples:

- TO_S: <wenna@mydomain.com>, <ruby@mydomain.com>,
- CC: <admin@mydomain.com>, <judy@mydomain.com>,
- FROM: <admin@mydomain.com>

The following rules must be used when entering these parameters:

- The characters "TO_S", "CC" and "FROM:" must be entered.
- A space and an opening angle bracket "<" must be the entered before each address.
- An ending angle bracket ">" must be entered after each address.
- A comma character "," must be entered after each address for the TO_S and CC addresses.
- Only one e-mail address may be used for the FROM entry, with no comma at the end.

Because of run-time mode and memory usage, a syntax check is not performed on the TM_MAIL TO_S, CC and FROM parameters.

STATUS and SFC_STATUS parameters

The execution condition codes returned by TM_MAIL can be classified as follows:

- W#16#0000: Operation of TM_MAIL was completed successfully
- W#16#7xxx: Status of TM_MAIL operation
- W#16#8xxx: An error in an internal call to a communication device or the mail server

The following table shows the execution condition codes of TM_MAIL with the exception of the error codes from internally called communication modules.

Table 13- 4 Condition codes

STATUS (W#16#...):	SFC_STATUS (W#16#...):	Description
0000	-	The TM_MAIL operation completed without error. This zero STATUS code does not guarantee that an e-mail was actually sent (See the first item in the note following this table).
7001		TM_MAIL is active (BUSY = 1).
7002	7002	TM_MAIL is active (BUSY = 1).
8xxx	xxxx	The TM_MAIL operation was completed with an error in the internal communication instruction calls. For more information about the SFC_STATUS parameter, see the descriptions of the STATUS parameter of the communication instructions.
8010	xxxx	Failed to connect: For more information about the SFC_STATUS parameter, see the STATUS parameter of the TCON instruction.
8011	xxxx	Error sending data: For more information about SFC_STATUS parameter ,see the STATUS parameter of the TSEND instruction.
8012	xxxx	Error while receiving data: For more information about the SFC_STATUS parameter, see the STATUS parameter descriptions of the TRCV instruction.
8013	xxxx	Failed to connect: For more information for evaluating the SFC_STATUS parameter, see the STATUS parameter descriptions of the TCON and TDISCON instructions.
8014	-	Failed to connect: You may have entered an incorrect mail server IP address (ADDR_MAIL_SERVER) or too little time (WATCH_DOG_TIME) for the connection. It is also possible that the CPU has no connection to the network or the CPU configuration is incorrect.
82xx, 84xx, 85xx	-	The error message comes from the mail server and corresponds to error number "8" of the SMTP protocol. See the second item in the note following this table.
8450	-	Operation does not run: Mailbox is not available; try again later.
8451	-	Operation aborted: Local error in processing, .try again later
8500	-	Command syntax error: The cause may be that the e-mail server does not support the LOGIN authentication process. Check the parameters of TM_MAIL. Try to send an e-mail without authentication. Try replacing the parameter USERNAME with an empty string.
8501	-	Syntax error: Incorrect parameter or argument; you may have typed an incorrect address in the TO_S or CC parameters.
8502	-	Command is unknown or not implemented: Check your entries, especially the parameter FROM. Perhaps this is incomplete and you have omitted the "@" or "." characters.
8535	-	SMTP authentication is incomplete. You may have entered an incorrect username or password.
8550	-	The mail server cannot be reached, or you have no access rights. You may have entered an incorrect username or password or of your mail server does not support log in access. Another cause of this error could be an erroneous entry of the domain name after the "@" character in the TO_S or CC parameters.
8552	-	Operation aborted: Exceeded the allocated memory size; try again later.
8554	-	Transmission failed: Try again later.

Note

Possible unreported e-mail transmission errors

- Incorrect entry of a recipient address does not generate a STATUS error for TM_MAIL. In this case, there is no guarantee that additional recipients (with correctly specified e-mail addresses), will receive the e-mail.
- More information on SMTP error codes can be found on the Internet or in the error documentation for the mail server. You can also read the last error message from the mail server. The error message in stored in buffer1 of the instance DB parameter for TM_MAIL.

Online and diagnostic tools 14

14.1 Status LEDs

The CPU and the I/O modules use LEDs to provide information about either the operational status of the module or the I/O.

Status LEDs on a CPU

The CPU provides the following status indicators:

- STOP/RUN
  - Solid yellow indicates STOP mode
  - Solid green indicates RUN mode
  - Flashing (alternating green and yellow) indicates that the CPU is in STARTUP mode
- ERROR
  - Flashing red indicates an error, such as an internal error in the CPU, a error with the memory card, or a configuration error (mismatched modules)
  - Solid red indicates defective hardware
- MAINT (Maintenance) flashes whenever you insert a memory card. The CPU then changes to STOP mode. After the CPU has changed to STOP mode, perform one of the following functions to initiate the evaluation of the memory card:
  - Change the CPU to RUN mode
  - Perform a memory reset (MRES)
  - Power-cycle the CPU

You can also use the LED instruction (Page 282) to determine the status of the LEDs.

Table 14- 1 Status LEDs for a CPU

Description	STOP/RUN Yellow / Green	ERROR Red	MAINT Yellow
Power is off	Off	Off	Off
Startup, self-test, or firmware update	Flashing (alternating yellow and green)	-	Off
Stop mode	On (yellow)	-	-
Run mode	On (green)	-	-
Remove the memory card	On (yellow)	-	Flashing
Error	On (either yellow or green)	Flashing	-
Maintenance requested	On (either yellow or green)	-	On

Description	STOP/RUN Yellow / Green	ERROR Red	MAINT Yellow
Defective hardware	On (yellow)	On	Off
LED test or defective CPU firmware	Flashing (alternating yellow and green)	Flashing	Flashing

The CPU also provides two LEDs that indicate the status of the PROFINET communications. Open the bottom terminal block cover to view the PROFINET LEDs.

- Link (green) turns on to indicate a successful connection
- Rx/Tx (yellow) turns on to indicate transmission activity

The CPU and each digital signal module (SM) provide an I/O Channel LED for each of the digital inputs and outputs. The I/O Channel (green) turns on or off to indicate the state of the individual input or output.

Status LEDs on an SM

In addition, each digital SM provides a DIAG LED that indicates the status of the module:

- Green indicates that the module is operational
- Red indicates that the module is defective or non-operational

Each analog SM provides an I/O Channel LED for each of the analog inputs and outputs.

- Green indicates that the channel has been configured and is active
- Red indicates an error condition of the individual analog input or output

In addition, each analog SM provides a DIAG LED that indicates the status of the module:

- Green indicates that the module is operational
- Red indicates that the module is defective or non-operational

The SM detects the presence or absence of power to the module (field-side power, if required).

Table 14- 2 Status LEDs for a signal module (SM)

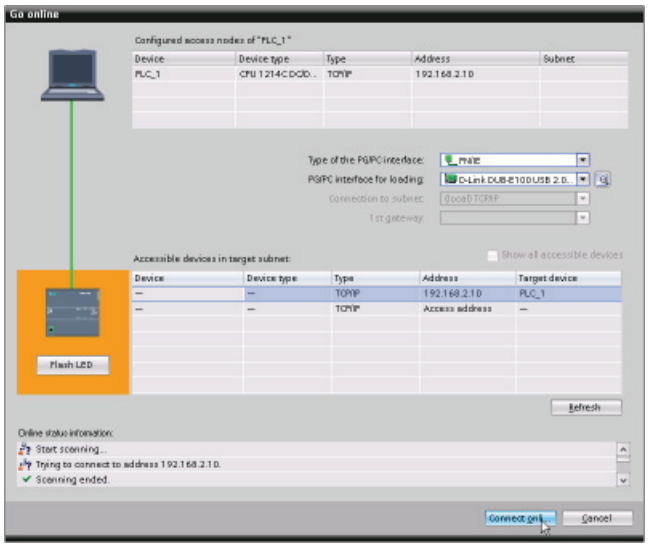
Description	DIAG (Red / Green)	I/O Channel (Red / Green)
Field-side power is off	Flashing red	Flashing red
Not configured or update in progress	Flashing green	Off
Module configured with no errors	On (green)	On (green)
Error condition	Flashing red	-
I/O error (with diagnostics enabled)	-	Flashing red
I/O error (with diagnostics disabled)	-	On (green)

14.2 Going online and connecting to a CPU

An online connection between the programming device and CPU is required for loading programs and project engineering data as well as for activities such as the following:

- Testing user programs
- Displaying and changing the operating mode of the CPU (Page 639)
- Displaying and setting the date and time of day of the CPU (Page 638)
- Displaying the module information
- Comparing and synchronizing (Page 641) offline to online program blocks
- Uploading and downloading program blocks
- Displaying diagnostics and the diagnostics buffer (Page 640)
- Using a watch table (Page 645) to test the user program by monitoring and modifying values
- Using a force table to force values in the CPU (Page 647)

To establish an online connection to a configured CPU, click the CPU from the Project Navigation tree and click the "Go online" button from the Project View:



If this is the first time to go online with this CPU, you must select the type of PG/PC interface and the specific PG/PC interface from the Go Online dialog before establishing an online connection to a CPU found on that interface.

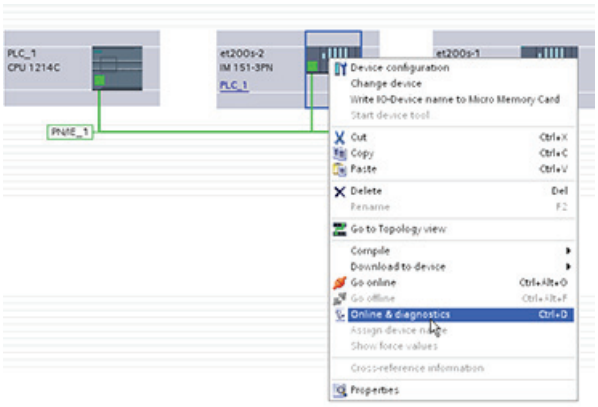
Your programming device is now connected to the CPU. The orange color frames indicate an online connection. You can now use the Online & diagnostics tools from the Project tree and the Online tools task card.

14.3 Assigning a name to a PROFINET IO device online

The devices on your PROFINET network must have an assigned name before you can connect with the CPU. Use the "Devices & networks" editor to assign names to your PROFINET devices if the devices have not already been assigned a name or if the name of the device is to be changed.

For each PROFINET IO device, you must assign the same name to that device in both the STEP 7 project and, using the "Online & diagnostics" tool, to the PROFINET IO device configuration memory (for example, an ET200 S interface module configuration memory). If a name is missing or does not match in either location, the PROFINET IO data exchange mode will not run.

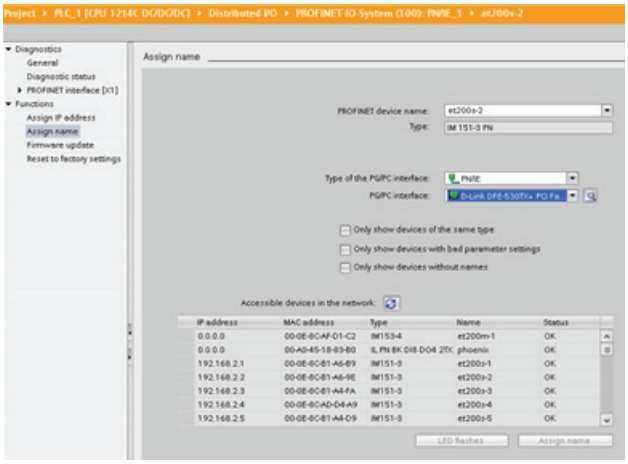
1. In the "Devices & networks" editor, right-click on the required PROFINET IO device, and select "Online & diagnostics".



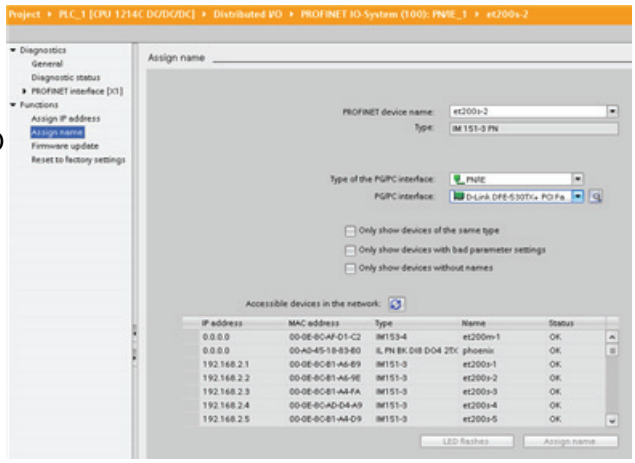
2. In the "Online & diagnostics" dialog, make the following menu selections:

- "Functions"
- "Assign name"

Click the "Accessible devices in the network" icon to display all of the PROFINET IO devices on the network.



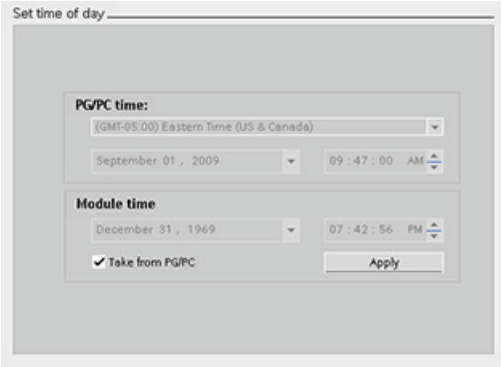
3. In the list that is displayed, click the required PROFINET IO device, and click the "Assign name" button to write the name to the PROFINET IO device configuration memory.



14.4 Setting the IP address and time of day

You can set the IP address and time of day in the online CPU. After accessing "Online & diagnostics" from the Project tree for an online CPU, you can display or change the IP address. You can also display or set the time and date parameters of the online CPU.

Refer to the section on the IP address for more information.



Note

This feature is available only for a CPU that either has only a MAC address (has not yet been assigned an IP address) or has been reset to factory settings.

14.5 Resetting to factory settings

You can reset an S7-1200 to its original factory settings under the following conditions:

- No memory card is inserted in the CPU.
- The CPU has an online connection.
- The CPU is in STOP mode.

Note

If the CPU is in RUN mode and you start the reset operation, you can place it in STOP mode after acknowledging a confirmation prompt.

Procedure

To reset a CPU to its factory settings, follow these steps:

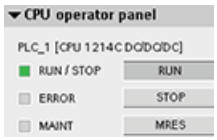
1. Open the Online and Diagnostics view of the CPU.
2. Select "Reset to factory settings" from the "Functions" folder.
3. Select the "Keep IP address" check box if you want to retain the IP address or the "Reset IP address" check box if you want to delete the IP address.
4. Click the "Reset" button.
5. Acknowledge the confirmation prompt with "OK".

Result

The module is switched to STOP mode if necessary, and it is reset to the factory settings:

- The work memory and internal load memory and all operand areas are cleared.
- All parameters are reset to their defaults.
- The diagnostics buffer is cleared.
- The time of day is reset.
- The IP address is retained or deleted based on the setting you made. (The MAC address is fixed and is never changed.)

14.6 CPU operator panel for the online CPU



The "CPU operator panel" displays the operating mode (STOP or RUN) of the online CPU. The panel also shows whether the CPU has an error or if values are being forced.

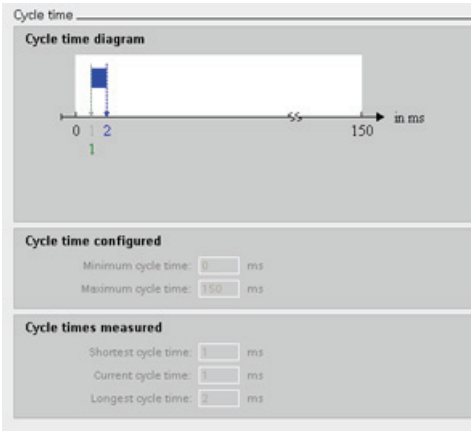
Use the CPU operating panel of the Online Tools task card to change the operating mode of an online CPU. The Online Tools task card is accessible whenever the CPU is online.

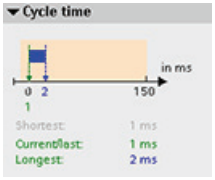
14.7 Monitoring the cycle time and memory usage

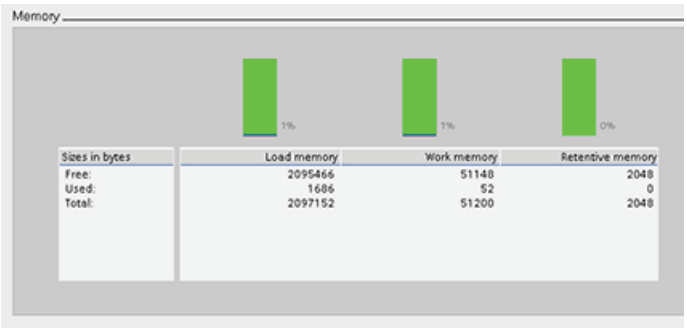
You can monitor the cycle time and memory usage of an online CPU.

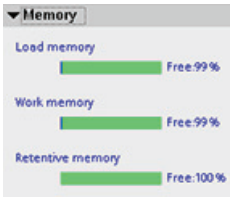
After connecting to the online CPU, open the Online tools task card to view the following measurements:

- Cycle time
- Memory usage





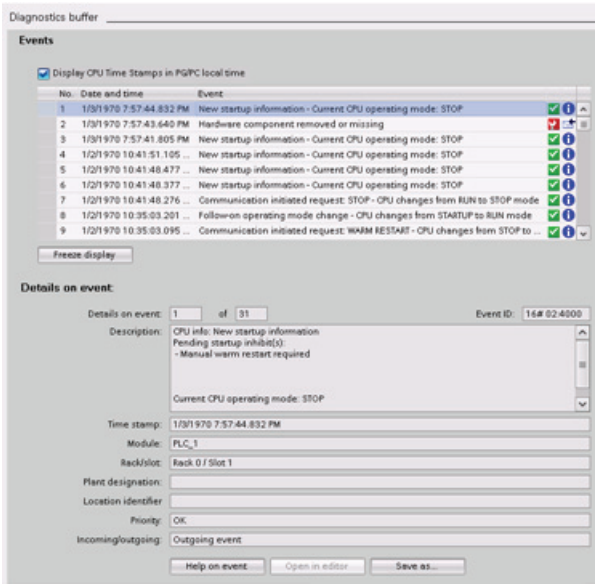




14.8 Displaying diagnostic events in the CPU

Use the diagnostics buffer to review the recent activity in the CPU. The diagnostics buffer is accessible from "Online & Diagnostics" for an online CPU in the Project tree. It contains the following entries:

- Diagnostic events
- Changes in the CPU operating mode (transitions to STOP or RUN mode)



The first entry contains the latest event. Each entry in the diagnostic buffer contains the date and time the event was logged, and a description.

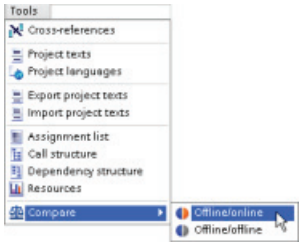
The maximum number of entries is dependent on the CPU. A maximum of 50 entries is supported.

Only the 10 most recent events in the diagnostic buffer are stored permanently. Resetting the CPU to the factory settings resets the diagnostic buffer by deleting the entries.

You can also use the GET_DIAG instruction (Page 285) to collect the diagnostic information.

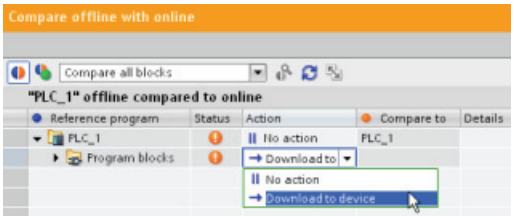
14.9 Comparing offline and online CPUs

You can compare the code blocks in an online CPU with the code blocks in your project. If the code blocks of your project do not match the code blocks of the online CPU, the "Compare" editor allows you to synchronize your project with the online CPU by downloading the code blocks of your project to the CPU, or by deleting blocks from the project that do not exist in the online CPU.



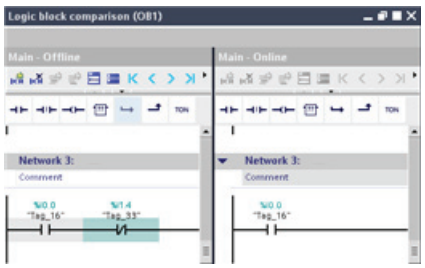
Select the CPU in your project.

Use the "Compare Offline/online" command to open the "Compare" editor. (Access the command either from the "Tools" menu or by right-clicking the CPU in your project.)



Click in the "Action" column for an object to select whether to delete the object, take no action, or download the object to the device.

Click the "Synchronize" button to load the code blocks.



Right-click an object in the "Compare to" column and select "Start detailed comparison" button to show the code blocks side-by-side.

The detailed comparison highlights the differences between the code blocks of online CPU and the code blocks of the CPU in your project.

14.10 Monitoring and modifying values in the CPU

STEP 7 provides online tools for monitoring the CPU:

- You can display or monitor the current values of the tags. The monitoring function does not change the program sequence. It presents you with information about the program sequence and the data of the program in the CPU.
- You can also use other functions to control the sequence and the data of the user program:
  - You can modify the value for the tags in the online CPU to see how the user program responds.
  - You can force a peripheral output (such as Q0.1:P or "Start":P) to a specific value.
  - You can enable outputs in STOP mode.

Note

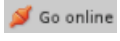
Always exercise caution when using control functions. These functions can seriously influence the execution of the user/system program.

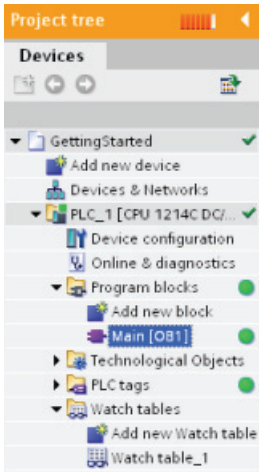
Table 14- 3 Online capabilities of the STEP 7 editors

Editor	Monitor	Modify	Force
Watch table	Yes	Yes	No
Force table	Yes	No	Yes
Program editor	Yes	Yes	No
Tag table	Yes	No	No
DB editor	Yes	No	No

14.10.1 Going online to monitor the values in the CPU

To monitor the tags, you must have an online connection to the CPU. Simply click the "Go online" button in the toolbar.

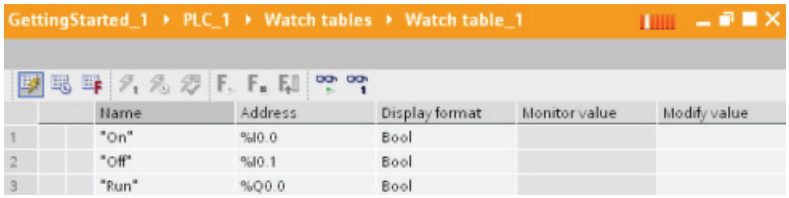




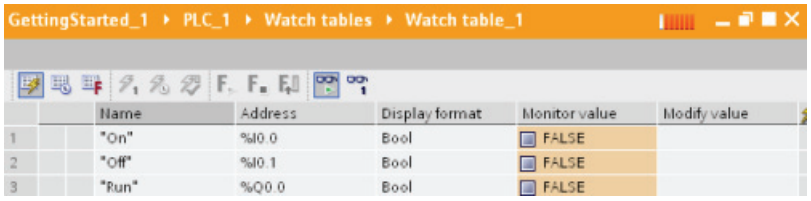
When you have connected to the CPU, STEP 7 turns the headers of the work areas orange.

The project tree displays a comparison of the offline project and the online CPU. A green circle means that the CPU and the project are synchronized, meaning that both have the same configuration and user program.

Tag tables show the tags. Watch tables can also show the tags, as well as direct addresses.



To monitor the execution of the user program and to display the values of the tags, click the "Monitor all" button in the toolbar.

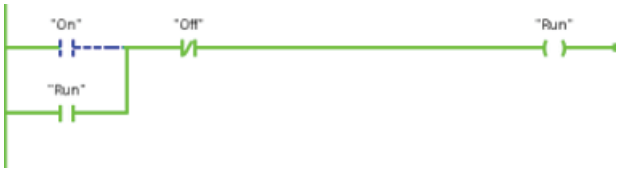


The "Monitor value" field shows the value for each tag.

14.10.2 Displaying status in the program editor

You can monitor the status of the tags in the LAD and FBD program editors. Use the editor bar to display the LAD editor. The editor bar allows you to change the view between the open editors without having to open or close the editors.

In the toolbar of the program editor, click the "Monitoring on/off" button to display the status of your user program.



The network in the program editor displays power flow in green.

You can also right-click on the instruction or parameter to modify the value for the instruction.

14.10.3 Capturing the online values of a DB to reset the start values

You can capture the current values being monitored in an online CPU to become the start values for a global DB.

- You must have an online connection to the CPU.
- The CPU must be in RUN mode.
- You must have opened the DB in STEP 7.

Use the "Show a snapshot of the monitored values" button to capture the current values of the selected tags in the DB. You can then copy these values into the "Start value" column of the DB.

1. In the DB editor, click the "Monitor all tags" button. The "Monitor value" column displays the current data values.
2. Click the "Show a snapshot of the monitored values" button to display the current values in the "Snapshot" column.
3. Click the "Monitor all" button to stop monitoring the data in the CPU.
4. Copy a value in the "Snapshot" column for a tag.
   - Select a value to be copied.
   - Right-click the selected value to display the context menu.
   - Select the "Copy" command.
5. Paste the copied value into the corresponding "Start value" column for the tag. (Right-click the cell and select "Paste" from the context menu.)
6. Save the project to configure the copied values as the new start values for the DB.
7. Compile and download the DB to the CPU. The DB uses the new start values after the CPU goes to RUN mode.

Note

The values that are shown in the "Monitor value" column are always copied from the CPU. STEP 7 does not check whether all values come from the same scan cycle of the CPU.

14.10.4 Using a watch table to monitor and modify values in the CPU

A watch table allows you to perform monitoring and control functions on data points as the CPU executes your program. These data points can be process image (I or Q), M, DB or physical inputs (I_:P), depending on the monitor or control function. You cannot accurately monitor the physical outputs (Q_:P) because the monitor function can only display the last value written from Q memory and does not read the actual value from the physical outputs.

The monitoring function does not change the program sequence. It presents you with information about the program sequence and the data of the program in the CPU.

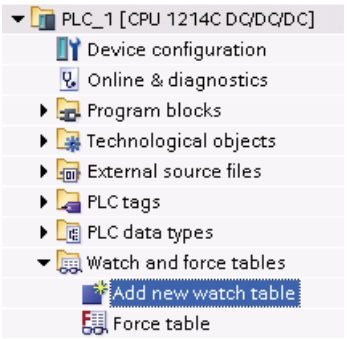
Control functions enable the user to control the sequence and the data of the program. Caution must be exercised when using control functions. These functions can seriously influence the execution of the user/system program. The three control functions are Modify, Force and Enable Outputs in STOP.

With the watch table, you can perform the following online functions:

- Monitoring the status of the tags
- Modifying values for the individual tags

You select when to monitor or modify the tag:

- Beginning of scan cycle: Reads or writes the value at the beginning of the scan cycle
- End of scan cycle: Reads or writes the value at the end of the scan cycle
- Switch to stop



To create a watch table:

1. Double-click "Add new watch table" to open a new watch table.
2. Enter the tag name to add a tag to the watch table.

The following options are available for monitoring tags:

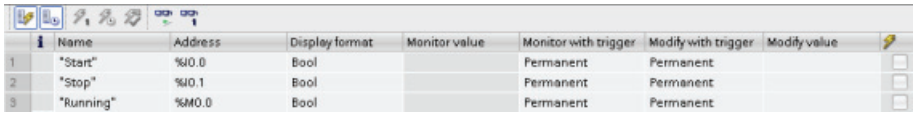
- Monitor all: This command starts the monitoring of the visible tags in the active watch table.
- Monitor now: This command starts the monitoring of the visible tags in the active watch table. The watch table monitors the tags immediately and once only.

The following options are available for modifying tags:

- "Modify to 0" sets the value of a selected address to "0".
- "Modify to 1" sets the value of a selected address to "1".
- "Modify now" immediately changes the value for the selected addresses for one scan cycle.
- "Modify with trigger" changes the values for the selected addresses.

  This function does not provide feedback to indicate that the selected addresses were actually modified. If feedback of the change is required, use the "Modify now" function.
- "Enable peripheral outputs" disables the command output disable and is available only when the CPU is in STOP mode.

To monitor the tags, you must have an online connection to the CPU.



	Name	Address	Display format	Monitor value	Monitor with trigger	Modify with trigger	Modify value
1	"Start"	%I0.0	Bool		Permanent	Permanent	
2	"Stop"	%I0.1	Bool		Permanent	Permanent	
3	"Running"	%M0.0	Bool		Permanent	Permanent	

The various functions can be selected using the buttons at the top of the watch table.

Enter the tag name to monitor and select a display format from the dropdown selection. With an online connection to the CPU, clicking the "Monitor" button displays the actual value of the data point in the "Monitor value" field.

14.10.4.1 Using a trigger when monitoring or modifying PLC tags

Triggering determines at what point in the scan cycle the selected address will be monitored or modified.

Table 14- 4 Types of triggers

Trigger	Description
Permanent	Continuously collects the data
At scan cycle start	Permanent: Continuously collects the data at the start of the scan cycle, after the CPU reads the inputs
	Once: Collects the data at the start of the scan cycle, after the CPU reads the inputs
At scan cycle end	Permanent: Continuously collects the data at the end of the scan cycle, before the CPU writes the outputs
	Once: Collects the data once at the end of the scan cycle, before the CPU writes the outputs
At transition to STOP	Permanent: Continuously collects data when the CPU transitions to STOP
	Once: Collects the data once after the CPU transitions to STOP

For modifying a PLC tag at a given trigger, select either the start or the end of cycle.

- Modifying an output: The best trigger event for modifying an output is at the end of the scan cycle, immediately before the CPU writes the outputs.

  Monitor the value of the outputs at the beginning of the scan cycle to determine what value is written to the physical outputs. Also, monitor the outputs before the CPU writes the values to the physical outputs in order to check program logic and to compare to the actual I/O behavior.

- Modifying an input: The best trigger event for modifying an input is at the start of the cycle, immediately after the CPU reads the inputs and before the user program uses the input values.

  If you are modifying inputs the start of the scan cycle, you should also monitor the value of the inputs at the end of the scan cycle to ensure that the value of the input at the end the scan cycle has not changed from the start of the scan cycle. If there is a difference in the values, your user program may be writing to an input in error.

To diagnose why the CPU might have gone to STOP, use the "Transition to STOP" trigger to capture the last process values.

14.10.4.2 Enabling outputs in STOP mode

The watch table allows you to write to the outputs when the CPU is in STOP mode. This functionality allows you to check the wiring of the outputs and verify that the wire connected to an output pin initiates a high or low signal to the terminal of the process device to which it is connected.

⚠ WARNING
Even though the CPU is in STOP mode, enabling a physical output can activate the process point to which it is connected.

You can change the state of the outputs in STOP mode when the outputs are enabled. If the outputs are disabled, you cannot modify the outputs in STOP mode.

- To enable the modification of the outputs in STOP, select the "Enable peripheral outputs" option of the "Modify" command of the "Online" menu, or by right-clicking the row of the Watch table.

  You cannot enable outputs in STOP mode if you have configured distributed I/O,. An error is returned when you try to do this.
- Setting the CPU to RUN mode disables "Enable peripheral outputs" option.
- If any inputs or outputs are forced, the CPU is not allowed to enable outputs while in STOP mode. The force function must first be cancelled.

14.10.5 Forcing values in the CPU

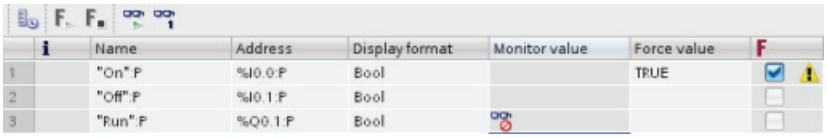
14.10.5.1 Using the force table

A force table provides a "force" function that overwrites the value for an input or output point to a specified value for the peripheral input or peripheral output address. The CPU applies this forced value to the input process image prior to the execution of the user program and to the output process image before the outputs are written to the modules.

Note

The force values are stored in the CPU and not in the force table.

You cannot force an input (or "I" address) or an output (or "Q" address). However, you can force a peripheral input or peripheral output. The force table automatically appends a ":P" to the address (for example: "On":P or "Run":P).



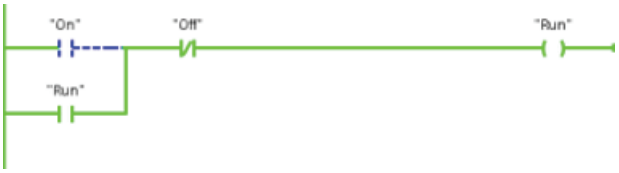
	i	Name	Address	Display format	Monitor value	Force value	F
1		"On":P	%I0.0:P	Bool		TRUE	☑ ⚠
2		"Off":P	%I0.1:P	Bool			☐
3		"Run":P	%Q0.1:P	Bool			☐

In the "Force value" cell, enter the value for the input or output to be forced. You can then use the check box in the "Force" column to enable forcing of the input or output.

Use the "Start or replace forcing" button to force the value of the tags in the force table. Click the "Stop forcing" button to reset the value of the tags.

In the force table, you can monitor the status of the forced value for an input. However, you cannot monitor the forced value of an output.

You can also view the status of the forced value in the program editor.



NOTICE

When an input or output is forced in a force table, the force actions become part of the project configuration. If you close STEP 7, the forced elements remain active in the CPU program until they are cleared. To clear these forced elements, you must use STEP 7 to connect with the online CPU and then use the force table to turn off or stop the force function for those elements.

14.10.5.2 Operation of the Force function

The CPU allows you to force input and output point(s) by specifying the physical input or output address (I_:P or Q_:P) in the watch table and then starting the force function.

In the program, reads of physical inputs are overwritten by the forced value. The program uses the forced value in processing. When the program writes a physical output, the output value is overwritten by the force value. The forced value appears at the physical output and is used by the process.

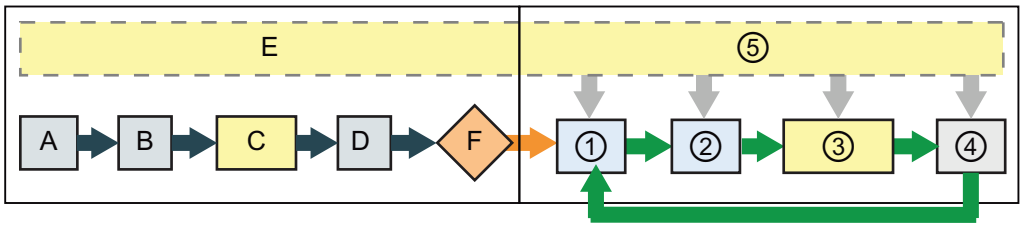
When an input or output is forced in the force table, the force actions become part of the user program. Even though the programming software has been closed, the force selections remain active in the operating CPU program until they are cleared by going online with the programming software and stopping the force function. Programs with forced points loaded on another CPU from a memory card will continue to force the points selected in the program.

If the CPU is executing the user program from a write-protected memory card, you cannot initiate or change the forcing of I/O from a watch table because you cannot override the values in the write-protected user program. Any attempt to force the write-protected values generates an error. If you use a memory card to transfer a user program, any forced elements on that memory card will be transferred to the CPU.

Note

Digital I/O points assigned to HSC, PWM, and PTO cannot be forced

The digital I/O points used by the high-speed counter (HSC), pulse-width modulation (PWM), and pulse-train output (PTO) devices are assigned during device configuration. When digital I/O point addresses are assigned to these devices, the values of the assigned I/O point addresses cannot be modified by the force function of the watch table.



Startup

- A The clearing of the I memory area is not affected by the Force function.
- B The initialization of the outputs values is not affected by the Force function.
- C During the execution of the startup OBs, the CPU applies the force value when the user program accesses the physical input.
- D The storing of interrupt events into the queue is not affected.
- E The enabling of the writing to the outputs is not affected.

RUN

- ① While writing Q memory to the physical outputs, the CPU applies the force value as the outputs are updated.
- ② When reading the physical inputs, the CPU applies the force values just prior to copying the inputs into I memory.
- ③ During the execution of the user program (program cycle OBs), the CPU applies the force value when the user program accesses the physical input or writes the physical output.
- ④ Handling of communication requests and self-test diagnostics are not affected by the Force function.
- ⑤ The processing of interrupts during any part of the scan cycle is not affected.

14.11 Downloading in RUN mode

The CPU supports "Download in RUN mode". This capability is intended to allow you to make small changes to a user program with minimal disturbance to the process being controlled by the program. However, implementing this capability also allows massive program changes that could be disruptive or even dangerous.

⚠ WARNING

When you download changes to the CPU in RUN mode, the changes immediately affect process operation. Changing the program in RUN mode can result in unexpected system operation, which could cause death or serious injury to personnel, and/or damage to equipment.

Only authorized personnel who understand the effects of RUN mode changes on system operation should perform a download in RUN mode.

The "Download in RUN mode" feature allows you to make changes to a program and download them to your CPU without switching to STOP mode:

- You can make minor changes to your current process without having to shut down (for example, change a parameter value).
- You can debug a program more quickly with this feature (for example, invert the logic for a normally open or normally closed switch).

You can make the following program block and tag changes and download them in RUN mode:

- Create, overwrite, and delete Functions (FC), Function Blocks (FB), and Tag tables.
- Create and delete Data Blocks (DB); however, DB structure changes cannot be overwritten. Initial DB values can be overwritten. You cannot download a web server DB (control or fragment) in RUN mode.
- Overwrite Organization Blocks (OB); however, you cannot create or delete OBs.

A maximum number of ten blocks can be downloaded in RUN mode at one time. If more than ten blocks are downloaded, the CPU must be placed into STOP mode.

If you download changes to a real process (as opposed to a simulated process, which you might do in the course of debugging a program), it is vital to think through the possible safety consequences to machines and machine operators before you download.

Note

If the CPU is in RUN mode and program changes have been made, the TIA Portal will always try to download in RUN first. If you do not want this to happen, you must put the CPU into STOP.

If the changes made are not supported in "Download in RUN", the TIA Portal will prompt the user that the CPU must go to STOP.

14.11.1 Prerequisites for "Download in RUN mode"

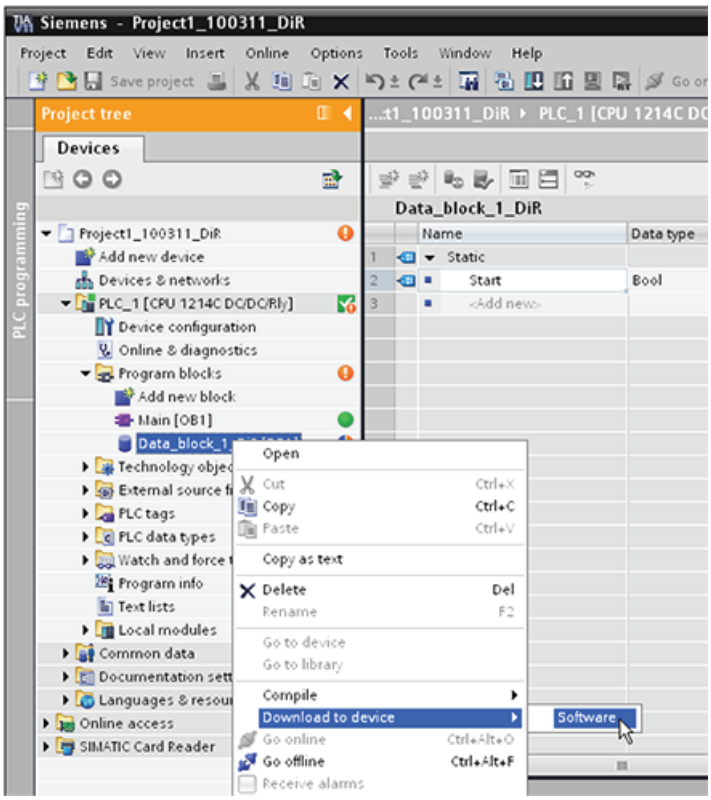
You cannot download your program changes to a CPU that is in RUN mode unless you have met these prerequisites:

- Your program must compile successfully.
- You must have successfully established communication between the programming device where you are running STEP 7 and the CPU.
- The firmware of the target CPU must support the "Download in RUN mode" feature.

14.11.2 Changing your program in RUN mode

To change the program in RUN mode, your must first ensure that the CPU supports "Download in RUN mode" and that the CPU is in RUN mode:

1. To download your program in RUN mode, select one of the following methods:
   - "Download to device" command from the "Online" menu
   - "Download to device" button in the toolbar
   - In the "Project tree", right-click "Program blocks" and select the "Download to device > Software" command.



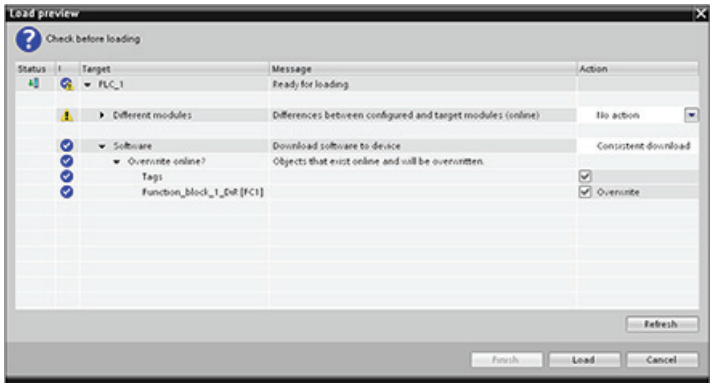
2. If the program compiles successfully, STEP 7 downloads the program to the CPU.

3. STEP 7 prompts you to load your program or cancel the operation.
4. If you click "Load", STEP 7 downloads the program to the CPU.

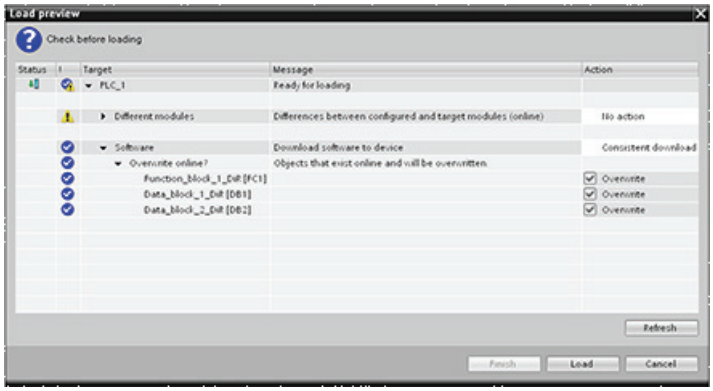
14.11.3 Downloading selected blocks

The focus is on the Program blocks folder, selection of blocks, or one single block.

1. If the user selects a single block for downloading from within the block editor, then the only option in the "Action" column is "Consistent download".
The user can expand the category line to be sure what blocks are to be loaded. In this example, a small change was made to the offline block, and no other blocks need to be loaded.



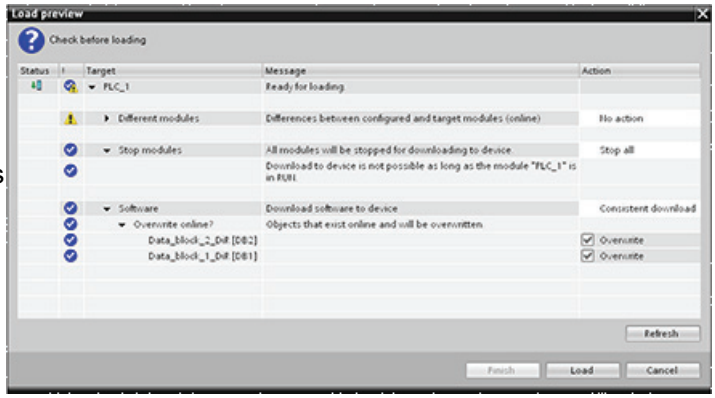
2. In this example, more than one block is needed for downloading.



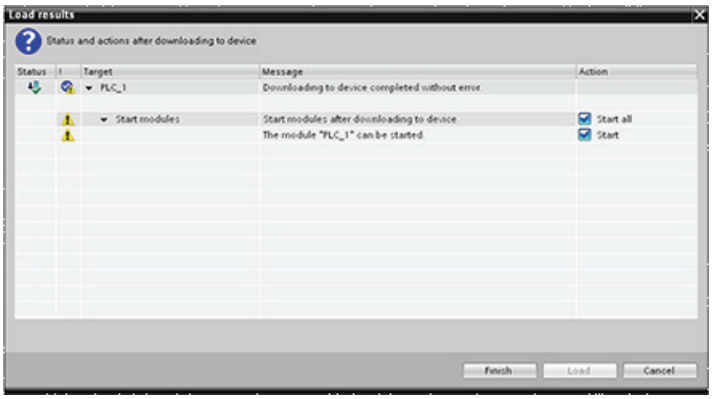
Note

A maximum number of ten blocks can be downloaded in RUN mode at one time. If more than ten blocks are downloaded, the CPU must be placed into STOP mode.

3. If the user attempts to download in RUN, but the system detects that this is not possible prior to the actual download, then the Stop modules category line appears in the dialog.

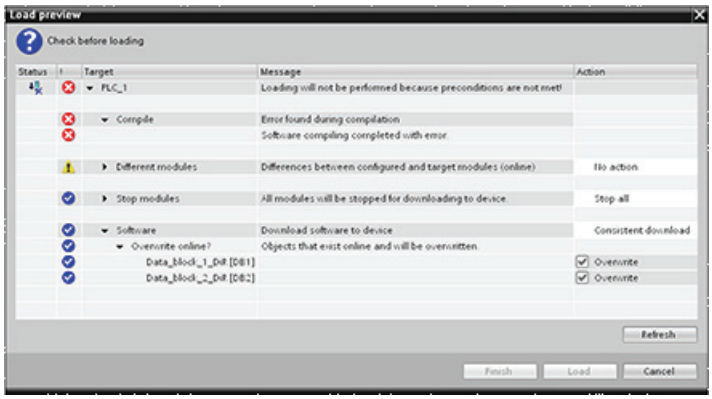


4. Click the "Load" button, and the "Load results" dialog appears. Click the "Finish" button to complete the download.

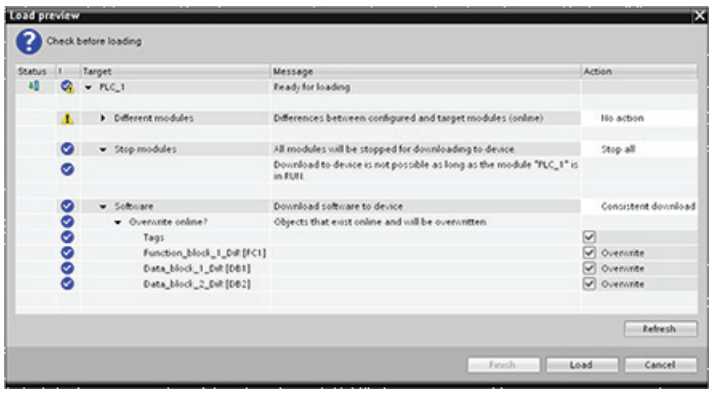


14.11.4 Downloading a single selected block with a compile error in another block

If the user attempts a consistent download with compile error in another block, then the dialog will indicate an error, and the load button will be disabled.

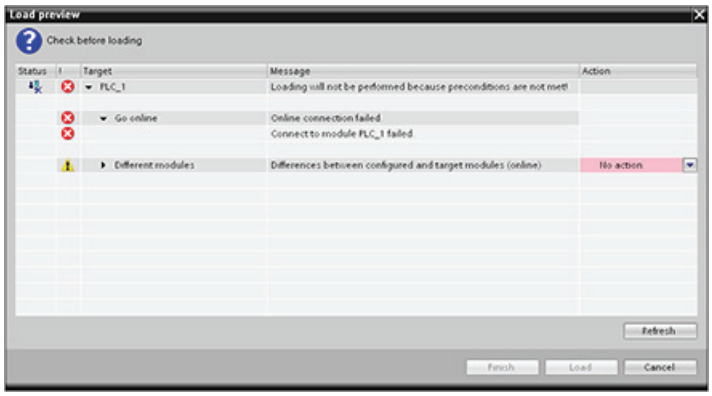


The user must correct the compile error in the other block. Then, the "Load" button becomes active.



14.11.5 System reaction if the download process fails

During the initial Download in RUN operation, if a network connection failure occurs, a "Load preview" dialog will result as shown in the figure below.



14.11.6 Downloading the program in RUN mode

Before downloading the program in RUN mode, consider the effect of a RUN-mode modification on the operation of the CPU for the following situations:

- If you deleted the control logic for an output, the CPU maintains the last state of the output until the next power cycle or transition to STOP mode.
- If you deleted a high-speed counter or pulse output functions which were running, the high-speed counter or pulse output continues to run until the next power cycle or transition to STOP mode.

- Any logic that is conditional on the state of the first scan bit will not be executed until the next power cycle or transition from STOP to RUN mode. The first scan bit is set only by the transition to RUN mode and is not affected by a download in RUN mode.
- The current values of data blocks (DB) and/or tags will not be overwritten.

Note

Before you can download your program in RUN mode, the CPU must support changes in RUN mode, the program must compile with no errors, and the communication between STEP 7, and the CPU must be error-free.

You can make the following changes in program blocks and tags and download them in RUN mode:

- Create, overwrite, and delete Functions (FC), Function Blocks (FB), and Tag tables.
- Create and delete Data Blocks (DB); however, DB structure changes cannot be overwritten. Initial DB values can be overwritten. You cannot download a web server DB (control or fragment) in RUN mode.
- Overwrite Organization Blocks (OB); however, you cannot create or delete OBs.

A maximum number of ten blocks can be downloaded in RUN mode at one time. If more than ten blocks are downloaded, the CPU must be placed into STOP mode.

Once a download is initiated, you cannot perform other tasks in STEP 7 until the download is complete.

Instructions that may fail due to "Download in RUN mode"

The following instructions may experience a temporary error when download in run changes are being activated in the CPU. The error occurs when the instruction is initiated while the CPU is preparing to activate the downloaded changes. During this time, the CPU suspends initiation of user-program access to the Load Memory, while it completes in-progress user-program access to Load Memory. This is done so that downloaded changes can be activated consistently.

Instruction	Response while Activation is Pending
DataLogCreate	STATUS = W#16#80C0, ERROR = TRUE
DataLogOpen	STATUS = W#16#80C0, ERROR = TRUE
DataLogWrite	STATUS = W#16#80C0, ERROR = TRUE
DataLogClose	STATUS = W#16#80C0, ERROR = TRUE
DataLogNewFile	STATUS = W#16#80C0, ERROR = TRUE
READ_DBL	RET_VAL = W#16#82C0
WRIT_DBL	RET_VAL = W#16#82C0
RTM	RET_VAL = 0x80C0

In all cases the RLO output from the instruction will be false when the error occurs. The error is temporary. If it occurs, the instruction should be retried later.

Note

You must not retry the operation in the current execution of the OB.

Technical specifications

A

A.1 General Technical Specifications

Standards compliance

The S7-1200 automation system complies with the following standards and test specifications. The test criteria for the S7-1200 automation system are based on these standards and test specifications.

CE approval

The S7-1200 Automation System satisfies requirements and safety related objectives according to the EC directives listed below, and conforms to the harmonized European standards (EN) for the programmable controllers listed in the Official Journals of the European Community.

- EC Directive 2006/95/EC (Low Voltage Directive) "Electrical Equipment Designed for Use within Certain Voltage Limits"
  - EN 61131-2:2007 Programmable controllers - Equipment requirements and tests
- EC Directive 2004/108/EC (EMC Directive) "Electromagnetic Compatibility"
  - Emission standard
    EN 61000-6-4:2007: Industrial Environment
  - Immunity standard
    EN 61000-6-2:2005: Industrial Environment
- EC Directive 94/9/EC (ATEX) "Equipment and Protective Systems Intended for Use in Potentially Explosive Atmosphere"
  - EN 60079-15:2005: Type of Protection 'n'

The CE Declaration of Conformity is held on file available to competent authorities at:

Siemens AG
IA AS RD ST PLC Amberg
Werner-von-Siemens-Str. 50
D92224 Amberg
Germany

cULus approval

Underwriters Laboratories Inc. complying with:

- Underwriters Laboratories, Inc.: UL 508 Listed (Industrial Control Equipment)
- Canadian Standards Association: CSA C22.2 Number 142 (Process Control Equipment)

NOTICE
The SIMATIC S7-1200 series meets the CSA standard. The cULus logo indicates that the S7-1200 has been examined and certified by Underwriters Laboratories (UL) to standards UL 508 and CSA 22.2 No. 142.

FM approval

Factory Mutual Research (FM)
Approval Standard Class Number 3600 and 3611
Approved for use in:
Class I, Division 2, Gas Group A, B, C, D, Temperature Class T4A Ta = 40° C
Class I, Zone 2, IIC, Temperature Class T4 Ta = 40° C
Canadian Class I, Zone 2 Installation per CEC 18-150

Note

The SM 1223 DI 8 x 120/230 VAC, DQ 8 x Relay (6ES7 223-1QH30-0XB0) signal module is approved for use in Class 1, Division 2, Gas Group A, B, C, D, Temperature Class T4 Ta = 40° C.

ATEX approval

EN 60079-0:2006: Explosive Atmospheres - General Requirements

EN 60079-15:2005: Electrical Apparatus for potentially explosive atmospheres;
Type of protection 'n'
II 3 G Ex nA II T4

The following special conditions for safe use of the S7-1200 must be followed:

- Install modules in a suitable enclosure providing a minimum degree of protection of IP54 according to EN 60529 and take into account the environmental conditions under which the equipment will be used.
- When the temperature under rated conditions exceeds 70° C at the cable entry point, or 80° C at the branching point of the conductors, the temperature specification of the selected cable should be in compliance with the actual measured temperature.
- Provisions should be made to prevent the rated voltage from being exceeded by transient disturbances of more than 40%.

C-Tick approval

The S7-1200 automation system satisfies requirements of standards to AS/NZS 2064 (Class A).

Korea Certification

The S7-1200 automation system satisfies the requirements of the Korean Certification (KC Mark). It has been defined as Class A Equipment and is intended for industrial applications and has not been considered for home use.

Maritime approval

The S7-1200 products are periodically submitted for special agency approvals related to specific markets and applications. Consult your local Siemens representative if you need additional information related to the latest listing of exact approvals by part number.

Classification societies:

- ABS (American Bureau of Shipping)
- BV (Bureau Veritas)
- DNV (Det Norske Veritas)
- GL (Germanischer Lloyd)
- LRS (Lloyds Register of Shipping)
- Class NK (Nippon Kaiji Kyokai)

Note

The CM 1242-5 (PROFIBUS Slave module), the CM 1243-5 (PROFIBUS Master module), and the CP 1242-7 (GPRS module) do not have Maritime approval.

Industrial environments

The S7-1200 automation system is designed for use in industrial environments.

Table A- 1 Industrial environments

Application field	Noise emission requirements	Noise immunity requirements
Industrial	EN 61000-6-4:2007	EN 61000-6-2:2005

Electromagnetic compatibility

Electromagnetic Compatibility (EMC) is the ability of an electrical device to operate as intended in an electromagnetic environment and to operate without emitting levels of electromagnetic interference (EMI) that may disturb other electrical devices in the vicinity.

Table A- 2 Immunity per EN 61000-6-2

Electromagnetic compatibility - Immunity per EN 61000-6-2	
EN 61000-4-2 Electrostatic discharge	8 kV air discharge to all surfaces 6 kV contact discharge to exposed conductive surfaces
EN 61000-4-3 Radiated, radio-frequency, electromagnetic field immunity test	80 to 1000 MHz, 10 V/m, 80% AM at 1 kHz 1.4 to 2.0 GHz, 3 V/m, 80% AM at 1 kHz 2.0 to 2.7 GHz, 1 V/m, 80% AM at 1 kHz
EN 61000-4-4 Fast transient bursts	2 kV, 5 kHz with coupling network to AC and DC system power 2 kV, 5 kHz with coupling clamp to I/O
EN 6100-4-5 Surge immunity	AC systems - 2 kV common mode, 1kV differential mode DC systems - 2 kV common mode, 1kV differential mode For DC systems (I/O signals, DC power systems) external protection is required.
EN 61000-4-6 Conducted disturbances	150 kHz to 80 MHz, 10 V RMS, 80% AM at 1kHz
EN 61000-4-11 Voltage dips	AC systems 0% for 1 cycle, 40% for 12 cycles and 70% for 30 cycles at 60 Hz

Table A- 3 Conducted and radiated emissions per EN 61000-6-4

Electromagnetic compatibility - Conducted and radiated emissions per EN 61000-6-4		
Conducted Emissions EN 55011, Class A, Group 1	0.15 MHz to 0.5 MHz	<79dB (µV) quasi-peak; <66 dB (µV) average
	0.5 MHz to 5 MHz	<73dB (µV) quasi-peak; <60 dB (µV) average
	5 MHz to 30 MHz	<73dB (µV) quasi-peak; <60 dB (µV) average
Radiated Emissions EN 55011, Class A, Group 1	30 MHz to 230 MHz	<40dB (µV/m) quasi-peak; measured at 10m
	230 MHz to 1 GHz	<47dB (µV/m) quasi-peak; measured at 10m

Environmental conditions

Table A- 4 Transport and storage

Environmental conditions - Transport and storage	
EN 60068-2-2, Test Bb, Dry heat and EN 60068-2-1, Test Ab, Cold	-40° C to +70° C
EN 60068-2-30, Test Db, Damp heat	25° C to 55° C, 95% humidity
EN 60068-2-14, Test Na, temperature shock	-40° C to +70° C, dwell time 3 hours, 5 cycles
EN 60068-2-32, Free fall	0.3 m, 5 times, product packaging
Atmospheric pressure	1080 to 660h Pa (corresponding to an altitude of -1000 to 3500m)

Table A- 5 Operating conditions

Environmental conditions - Operating	
Ambient temperature range (Inlet Air 25 mm below unit)	0° C to 55° C horizontal mounting 0° C to 45° C vertical mounting 95% non-condensing humidity
Atmospheric pressure	1080 to 795 hPa (corresponding to an altitude of -1000 to 2000m)
Concentration of contaminants	$S0_2$: < 0.5 ppm; H_2S: < 0.1 ppm; RH < 60% non-condensing
EN 60068-2-14, Test Nb, temperature change	5° C to 55° C, 3° C/minute
EN 60068-2-27 Mechanical shock	15 G, 11 ms pulse, 6 shocks in each of 3 axis
EN 60068-2-6 Sinusoidal vibration	DIN rail mount: 3.5 mm from 5-9 Hz, 1G from 9 - 150 Hz Panel Mount: 7.0 mm from 5-9 Hz, 2G from 9 to 150 Hz 10 sweeps each axis, 1 octave per minute

Table A- 6 High potential isolation test

High potential isolation test	
24 V/5 V nominal circuits	520 VDC (type test of optical isolation boundaries)
115/230 V circuits to ground	1,500 VAC routine test/1950 VDC type test
115/230 V circuits to 115/230 V circuits	1,500 VAC routine test/1950 VDC type test
115 V/230V circuits to 24 V/5 V circuits	1,500 VAC routine test/3250 VDC type test

Protection class

- Protection Class II according to EN 61131-2 (Protective conductor not required)

Degree of protection

- IP20 Mechanical Protection, EN 60529
- Protects against finger contact with high voltage as tested by standard probe. External protection required for dust, dirt, water and foreign objects of < 12.5mm in diameter.

Rated voltages

Table A- 7 Rated voltages

Rated voltage	Tolerance
24 VDC	20.4 VDC to 28.8 VDC
120/230 VAC	85 VAC to 264 VAC, 47 to 63 Hz

NOTICE
When a mechanical contact turns on output power to the S7-1200 CPU, or any digital signal module, it sends a "1" signal to the digital outputs for approximately 50 microseconds. This could cause unexpected machine or process operation which could result in death or serious injury to personnel and/or damage to equipment. You must plan for this, especially if you are using devices which respond to short duration pulses.

Reverse voltage protection

Reverse voltage protection circuitry is provided on each terminal pair of +24 VDC power or user input power for CPUs, signal modules (SMs), and signal boards (SBs). It is still possible to damage the system by wiring different terminal pairs in opposite polarities.

Some of the 24 VDC power input ports in the S7-1200 system are interconnected, with a common logic circuit connecting multiple M terminals. For example, the following circuits are interconnected when designated as "not isolated" in the data sheets: the 24 VDC power supply of the CPU, the power input for the relay coil of an SM, or the power supply for a non-isolated analog input. All non-isolated M terminals must connect to the same external reference potential.

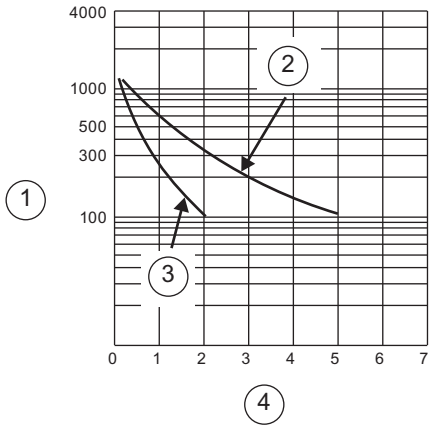
⚠ WARNING
Connecting non-isolated M terminals to different reference potentials will cause unintended current flows that may cause damage or unpredictable operation in the PLC and any connected equipment. Failure to comply with these guidelines could cause damage or unpredictable operation which could result in death or serve personal injury and/or property damage. Always ensure that all non-isolated M terminals in an S7-1200 system are connected to the same reference potential.

DC Outputs

Short circuit protection circuitry is not provided for DC outputs on CPUs, signal modules (SMs) and signal boards (SBs).

Relay electrical service life

The typical performance data supplied by relay vendors is shown below. Actual performance may vary depending upon your specific application. An external protection circuit that is adapted to the load will enhance the service life of the contacts.



① Service life (x 10^3 operations)

② 250 VAC resistive load
30 VDC resistive load

③ 250 VAC inductive load (p.f=0.4)
30 VDC inductive load (L/R=7ms)

④ Rated Operating Current (A)

A.2 CPU 1211C

A.2.1 General specifications and features

Table A- 8 General specifications

Technical data	CPU 1211C AC/DC/Relay	CPU 1211C DC/DC/Relay	CPU 1211C DC/DC/DC
Order number	6ES7 211-1BD30-0XB0	6ES7 211-1HD30-0XB0	6ES7 211-1AD30-0XB0
Dimensions W x H x D (mm)	90 x 100 x 75	90 x 100 x 75	90 x 100 x 75
Weight	420 grams	380 grams	370 grams
Power dissipation	10 W	8 W	8 W
Current available (CM bus)	750 mA max. (5 VDC)	750 mA max. (5 VDC)	750 mA max. (5 VDC)
Current available (24 VDC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24VDC)	4 mA/input used	4 mA/input used	4 mA/input used

Table A- 9 CPU features

Technical data		Description
User memory[1]	Work	25 Kbytes
	Load	1 Mbytes
	Retentive	2 Kbytes
On-board digital I/O		6 inputs/4 outputs
On-board analog I/O		2 inputs
Process image size		1024 bytes of inputs (I) /1024 bytes of outputs (Q)
Bit memory (M)		4096 bytes
Temporary (local) memory		• 16 Kbytes for startup and program cycle (including associated FBs and FCs) • 4 Kbytes for standard interrupt events including FBs and FCs • 4 Kbytes for error interrupt events including FBs and FCs
Signal modules expansion		none
Signal board expansion		1 SB max.
Communication module expansion		3 CMs max.
High-speed counters		3 total • Single phase: 3 at 100 kHz • Quadrature phase: 3 at 80 kHz
Pulse outputs [2]		2
Pulse catch inputs		6
Time delay / cyclic interrupts		4 total with 1 ms resolution
Edge interrupts		6 rising and 6 falling (10 and 10 with optional signal board)
Memory card		SIMATIC Memory Card (optional)
Real time clock accuracy		+/- 60 seconds/month
Real time clock retention time		10 days typ./6 days min. at 40°C (maintenance-free Super Capacitor)

[1] The size of the user program, data, and configuration is limited by the available load memory and work memory in the CPU. There is no specific limit to the number of OB, FC, FB and DB blocks supported or to the size of a particular block; the only limit is due to overall memory size.

[2] For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

Table A- 10 Performance

Type of instruction	Execution speed
Boolean	0.1 µs/instruction
Move Word	12 µs/instruction
Real math	18 µs/instruction

Table A- 11 Blocks, timers and counters supported by S7-1200

Element		Description
Blocks	Type	OB, FB, FC, DB
	Size	25 Kbytes (CPU 1211C and CPU 1212C) 50 Kbytes (CPU 1214C)
	Quantity	Up to 1024 blocks total (OBs + FBs + FCs + DBs)
	Address range for FBs, FCs, and DBs	1 to 65535 (such as FB 1 to FB 65535)
	Nesting depth	16 from the program cycle or start up OB; 4 from the time delay interrupt, time-of-day interrupt, cyclic interrupt, hardware interrupt, time error interrupt, or diagnostic error interrupt OB
	Monitoring	Status of 2 code blocks can be monitored simultaneously
OBs	Program cycle	Multiple: OB 1, OB 200 to OB 65535
	Startup	Multiple: OB 100, OB 200 to OB 65535
	Time-delay interrupts and cyclic interrupts	4[1] (1 per event): OB 200 to OB 65535
	Hardware interrupts (edges and HSC)	50 (1 per event): OB 200 to OB 65535
	Time error interrupts	1: OB 80
	Diagnostic error interrupts	1: OB 82
Timers	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, 16 bytes per timer
Counters	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, size dependent upon count type • SInt, USInt: 3 bytes • Int, UInt: 6 bytes • DInt, UDInt: 12 bytes

[1] Time-delay and cyclic interrupts use the same resources in the CPU. You can have only a total of 4 of these interrupts (time-delay plus cyclic interrupts). You cannot have 4 time-delay interrupts and 4 cyclic interrupts.

Table A- 12 Communication

Technical data	Description
Number of ports	1
Type	Ethernet
HMI device[1]	3
Programming device (PG)	1
Connections	• 8 for Open User Communication (active or passive): TSEND_C, TRCV_C, TCON, TDISCON, TSEND, and TRCV • 3 for server GET/PUT (CPU-to-CPU) S7 communication • 8 for client GET/PUT (CPU-to-CPU) S7 communication

Technical data	Description
Data rates	10/100 Mb/s
Isolation (external signal to PLC logic)	Transformer isolated, 1500 VDC
Cable type	CAT5e shielded

[1] The CPU provides dedicated HMI connections to support up to 3 HMI devices. (You can have up to 2 SIMATIC Comfort panels.) The total number of HMI is affected by the types of HMI panels in your configuration. For example, you could have up to three SIMATIC Basic panels connected to your CPU, or you could have up to two SIMATIC Comfort panels with one additional Basic panel.

Table A- 13 Power supply

Technical data		CPU 1211C AC/DC/Relay	CPU 1211C DC/DC/Relay	CPU 1211C DC/DC/DC
Voltage range		85 to 264 VAC	20.4 to 28.8 VDC	20.4 to 28.8 VDC
Line frequency		47 to 63 Hz	--	--
Input current	CPU only at max. load	60 mA at 120 VAC 30 mA at 240 VAC	300 mA at 24 VDC	300 mA at 24 VDC
	CPU with all expansion accessories at max. load	180 mA at 120 VAC 90 mA at 240 VAC	900 mA at 24 VDC	900 mA at 24 VDC
Inrush current (max.)		20 A at 264 VAC	12 A at 28.8 VDC	12 A at 28.8 VDC
Isolation (input power to logic)		1500 VAC	Not isolated	Not isolated
Ground leakage, AC line to functional earth		0.5 mA max.	--	--
Hold up time (loss of power)		20 ms at 120 VAC 80 ms at 240 VAC	10 ms at 24 VDC	10 ms at 24 VDC
Internal fuse, not user replaceable		3 A, 250 V, slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow

Table A- 14 Sensor power

Technical data	CPU 1211C AC/DC/Relay	CPU 1211C DC/DC/Relay	CPU 1211C DC/DC/DC
Voltage range	20.4 to 28.8 VDC	L+ minus 4 VDC min.	L+ minus 4 VDC min.
Output current rating (max.)	300 mA (short circuit protected)	300 mA (short circuit protected)	300 mA (short circuit protected)
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	Same as input line	Same as input line
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated

A.2.2 Digital inputs and outputs

Table A- 15 Digital inputs

Technical data	CPU 1211C AC/DC/Relay, DC/DC/Relay, and DC/DC/DC
Number of inputs	6
Type	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.
Logic 1 signal (min.)	15 VDC at 2.5 mA
Logic 0 signal (max.)	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute
Isolation groups	1
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 VDC)	Single phase: 100 KHz Quadrature phase: 80 KHz
Number of inputs on simultaneously	6
Cable length (meters)	500 m shielded, 300 m unshielded, 50 m shielded for HSC inputs

Table A- 16 Digital outputs

Technical data	CPU 1211C AC/DC/Relay and DC/DC/Relay	CPU 1211C DC/DC/DC
Number of outputs	4	4
Type	Relay, dry contact	Solid state - MOSFET (sourcing)
Voltage range	5 to 30 VDC or 5 to 250 VAC	20.4 to 28.8 VDC
Logic 1 signal at max. current	--	20 VDC min.
Logic 0 signal with 10 KΩ load	--	0.1 VDC max.
Current (max.)	2.0 A	0.5 A
Lamp load	30 W DC / 200 W AC	5 W
ON state resistance	0.2 Ω max. when new	0.6 Ω max.
Leakage current per point	--	10 µA max.
Surge current	7 A with contacts closed	8 A for 100 ms max.
Overload protection	No	No
Isolation (field side to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)	500 VAC for 1 minute
Isolation resistance	100 MΩ min. when new	--
Isolation between open contacts	750 VAC for 1 minute	--
Isolation groups	1	1
Inductive clamp voltage	--	L+ minus 48 VDC, 1 W dissipation

Technical data	CPU 1211C AC/DC/Relay and DC/DC/Relay	CPU 1211C DC/DC/DC
Maximum relay switching frequency	1 Hz	--
Switching delay (Qa.0 to Qa.3)	10 ms max.	1.0 µs max., off to on 3.0 µs max., on to off
Pulse Train Output rate (Qa.0 and Qa.2)	Not recommended [1]	100 KHz max., 2 Hz min.[2]
Lifetime mechanical (no load)	10,000,000 open/close cycles	--
Lifetime contacts at rated load	100,000 open/close cycles	--
Behavior on RUN to STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	4	4
Cable length (meters)	500 m shielded, 150 m unshielded	500 m shielded, 150 m unshielded

1 For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

2 Depending on your pulse receiver and cable, an additional load resistor (at least 10% of rated current) may improve pulse signal quality and noise immunity.

A.2.3 Analog inputs

Table A- 17 Analog inputs

Technical data	Description
Number of inputs	2
Type	Voltage (single-ended)
Range	0 to 10 V
Full-scale range (data word)	0 to 27648
Overshoot range (data word)	27,649 to 32,511 See the table for measurement ranges of the analog inputs for voltage (Page 669).
Overflow (data word)	32,512 to 32,767
Resolution	10 bits
Maximum withstand voltage	35 VDC
Smoothing	None, Weak, Medium, or Strong See the table for step response (ms) for the analog inputs of the CPU (Page 669).
Noise rejection	10, 50, or 60 Hz
Impedance	≥100 KΩ
Isolation (field side to logic)	None
Accuracy (25°C / 0 to 55°C)	3.0% / 3.5% of full-scale
Cable length (meters)	100 m, shielded twisted pair

A.2.3.1 Step response of the built-in analog inputs of the CPU

Table A- 18 Step Response (ms), 0V to 10V measured at 95%

Smoothing selection (sample averaging)	Rejection frequency (Integration time)		
	60 Hz	50 Hz	10 Hz
None (1 cycle): No averaging	63 ms	65 ms	130 ms
Weak (4 cycles): 4 samples	84 ms	93 ms	340 ms
Medium (16 cycles): 16 samples	221 ms	258 ms	1210 ms
Strong (32 cycles): 32 samples	424 ms	499 ms	2410 ms
Sample time	**4.17 ms**	**5 ms**	**25 ms**

A.2.3.2 Sample time for the built-in analog ports of the CPU

Table A- 19 Sample time for built-in analog inputs of the CPU

Rejection frequency(Integration time selection)	Sample time
60 Hz(16.6 ms)	4.17 ms
50 Hz (20 ms)	5 ms
10 Hz (100 ms)	25 ms

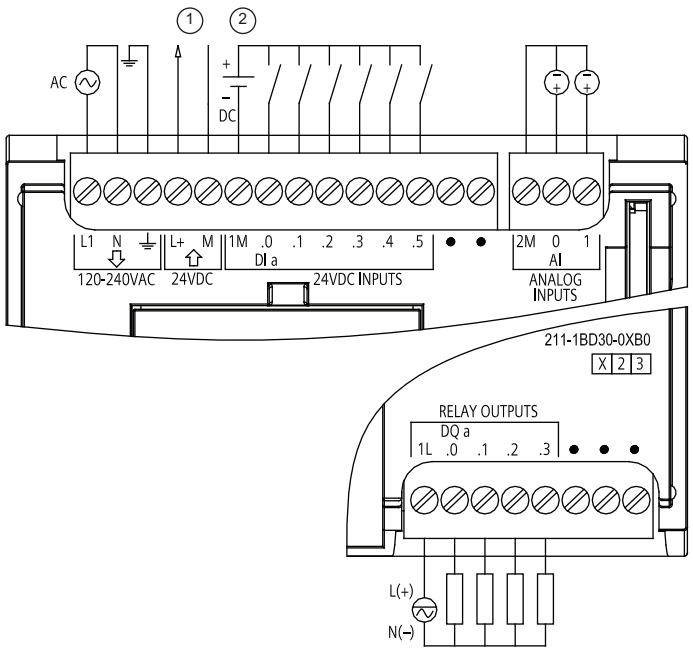
A.2.3.3 Measurement ranges of the analog inputs for voltage

Table A- 20 Analog input representation for voltage

System		Voltage Measuring Range					
Decimal	Hexadecimal	±10 V	±5 V	±2.5 V		0 to 10 V	
32767	7FFF	11.851 V	5.926 V	2.963 V	Overflow	11.851 V	Overflow
32512	7F00						
32511	7EFF	11.759 V	5.879 V	2.940 V	Overshoot range	11.759 V	Overshoot range
27649	6C01						
27648	6C00	10 V	5 V	2.5 V	Rated range	10 V	Rated range
20736	5100	7.5 V	3.75 V	1.875 V		7.5 V	
1	1	361.7 µV	180.8 µV	90.4 µV		361.7 µV	
0	0	0 V	0 V	0 V		0 V	
-1	FFFF					Negative values are not supported	
-20736	AF00	-7.5 V	-3.75 V	-1.875 V			
-27648	9400	-10 V	-5 V	-2.5 V			
-27649	93FF				Undershoot range		
-32512	8100	-11.759 V	-5.879 V	-2.940 V			
-32513	80FF				Underflow		
-32768	8000	-11.851 V	-5.926 V	-2.963 V			

A.2.4 Wiring diagrams

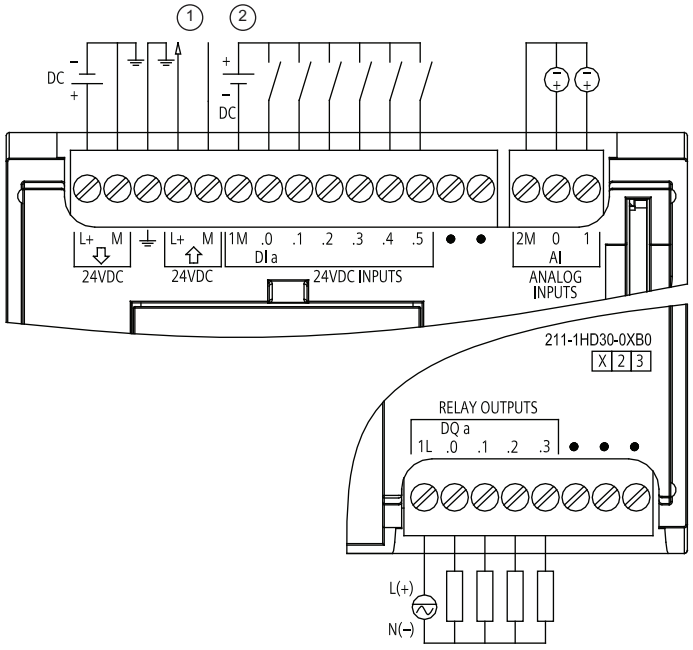
Table A- 21 CPU 1211C AC/DC/Relay (6ES7 211-1BD30-0XB0)



① 24 VDC Sensor Power Out
For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).
For sourcing inputs, connect "+" to "M".

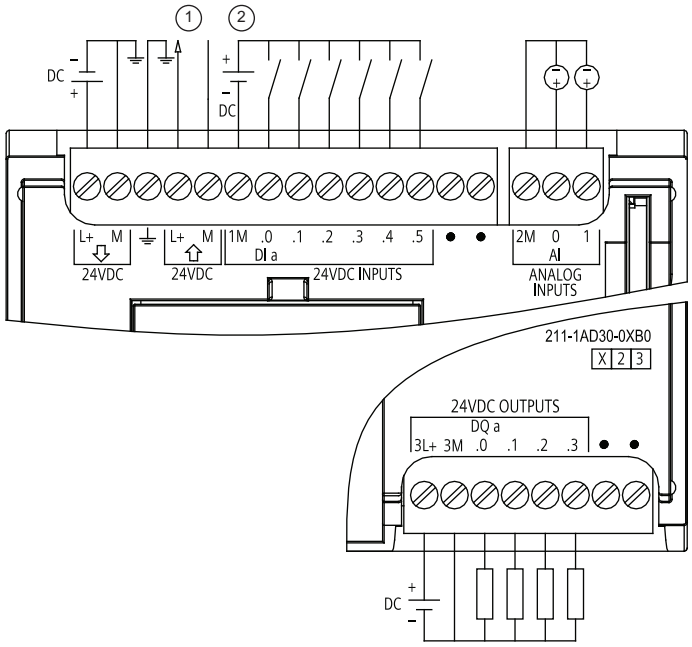
Table A- 22 CPU 1211C DC/DC/Relay (6ES7 211-1HD30-0XB0)



① 24 VDC Sensor Power Out
For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).
For sourcing inputs, connect "+" to "M".

Table A- 23 CPU 1211C DC/DC/DC (6ES7 211-1AD30-0XB0)



① 24 VDC Sensor Power Out

For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).

For sourcing inputs, connect "+" to "M".

Note

Unused analog inputs should be shorted.

A.3 CPU 1212C

A.3.1 General specifications and features

Table A- 24 General

Technical data	CPU 1212C AC/DC/Relay	CPU 1212C DC/DC/Relay	CPU 1212C DC/DC/DC
Order number	6ES7 212-1BD30-0XB0	6ES7 212-1HD30-0XB0	6ES7 212-1AD30-0XB0
Dimensions W x H x D (mm)	90 x 100 x 75	90 x 100 x 75	90 x 100 x 75
Weight	425 grams	385 grams	370 grams
Power dissipation	11 W	9 W	9 W
Current available (SM and CM bus)	1000 mA max. (5 VDC)	1000 mA max. (5 VDC)	1000 mA max. (5 VDC)

Technical data	CPU 1212C AC/DC/Relay	CPU 1212C DC/DC/Relay	CPU 1212C DC/DC/DC
Current available (24 VDC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 VDC)	4 mA/input used	4 mA/input used	4 mA/input used

Table A- 25 CPU features

Technical data		Description
User memory[1]	Work	25 Kbytes
	Load	1 Mbytes
	Retentive	2 Kbytes
On-board digital I/O		8 inputs/6 outputs
On-board analog I/O		2 inputs
Process image size		1024 bytes of inputs (I)/1024 bytes of outputs (Q)
Bit memory (M)		4096 bytes
Temporary (local) memory		• 16 Kbytes for startup and program cycle (including associated FBs and FCs) • 4 Kbytes for standard interrupt events including FBs and FCs • 4 Kbytes for error interrupt events including FBs and FCs
Signal modules expansion		2 SMs max.
Signal board expansion		1 SB max.
Communication module expansion		3 CMs max.
High-speed counters		4 total • Single phase: 3 at 100 kHz and 1 at 30 kHz clock rate • Quadrature phase: 3 at 80 kHz and 1 at 20 kHz clock rate
Pulse outputs [2]		2
Pulse catch inputs		8
Time delay / cyclic interrupts		4 total with 1 ms resolution
Edge interrupts		8 rising and 8 falling (12 and 12 with optional signal board)
Memory card		SIMATIC Memory Card (optional)
Real time clock accuracy		+/- 60 seconds/month
Real time clock retention time		10 days typ./6 days min. at 40°C (maintenance-free Super Capacitor)

1 The size of the user program, data, and configuration is limited by the available load memory and work memory in the CPU. There is no specific limit to the number of OB, FC, FB and DB blocks supported or to the size of a particular block; the only limit is due to overall memory size.

2 For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

Table A- 26 Performance

Type of instruction	Execution speed
Boolean	0.1 µs/instruction
Move Word	12 µs/instruction
Real math	18 µs/instruction

Table A- 27 Blocks, timers and counters supported by S7-1200

Element		Description
Blocks	Type	OB, FB, FC, DB
	Size	25 Kbytes (CPU 1211C and CPU 1212C) 50 Kbytes (CPU 1214C)
	Quantity	Up to 1024 blocks total (OBs + FBs + FCs + DBs)
	Address range for FBs, FCs, and DBs	1 to 65535 (such as FB 1 to FB 65535)
	Nesting depth	16 from the program cycle or start up OB; 4 from the time delay interrupt, time-of-day interrupt, cyclic interrupt, hardware interrupt, time error interrupt, or diagnostic error interrupt OB
	Monitoring	Status of 2 code blocks can be monitored simultaneously
OBs	Program cycle	Multiple: OB 1, OB 200 to OB 65535
	Startup	Multiple: OB 100, OB 200 to OB 65535
	Time-delay interrupts and cyclic interrupts	4[1] (1 per event): OB 200 to OB 65535
	Hardware interrupts (edges and HSC)	50 (1 per event): OB 200 to OB 65535
	Time error interrupts	1: OB 80
	Diagnostic error interrupts	1: OB 82
Timers	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, 16 bytes per timer
Counters	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, size dependent upon count type • SInt, USInt: 3 bytes • Int, UInt: 6 bytes • DInt, UDInt: 12 bytes

[1] Time-delay and cyclic interrupts use the same resources in the CPU. You can have only a total of 4 of these interrupts (time-delay plus cyclic interrupts). You cannot have 4 time-delay interrupts and 4 cyclic interrupts.

Table A- 28 Communication

Technical data	Description
Number of ports	1
Type	Ethernet
HMI device[1]	3
Programming device (PG)	1
Connections	• 8 for Open User Communication (active or passive): TSEND_C, TRCV_C, TCON, TDISCON, TSEND, and TRCV • 3 for server GET/PUT (CPU-to-CPU) S7 communication • 8 for client GET/PUT (CPU-to-CPU) S7 communication
Data rates	10/100 Mb/s
Isolation (external signal to PLC logic)	Transformer isolated, 1500 VDC
Cable type	CAT5e shielded

[1] The CPU provides dedicated HMI connections to support up to 3 HMI devices. (You can have up to 2 SIMATIC Comfort panels.) The total number of HMI is affected by the types of HMI panels in your configuration. For example, you could have up to three SIMATIC Basic panels connected to your CPU, or you could have up to two SIMATIC Comfort panels with one additional Basic panel.

Table A- 29 Power supply

Technical data		CPU 1212C AC/DC/Relay	CPU 1212C DC/DC/Relay	CPU 1212C DC/DC/DC
Voltage range		85 to 264 VAC	20.4 to 28.8 VDC	20.4 to 28.8 VDC
Line frequency		47 to 63 Hz	--	--
Input current (max. load)	CPU only	80 mA at 120 VAC 40 mA at 240 VAC	400 mA at 24 VDC	400 mA at 24 VDC
	CPU with all expansion accessories	240 mA at 120 VAC 120 mA at 240 VAC	1200 mA at 24 VDC	1200 mA at 24 VDC
Inrush current (max.)		20 A at 264 VAC	12 A at 28.8 VDC	12 A at 28.8 VDC
Isolation (input power to logic)		1500 VAC	Not isolated	Not isolated
Ground leakage, AC line to functional earth		0.5 mA max.	--	--
Hold up time (loss of power)		20 ms at 120 VAC 80 ms at 240 VAC	10 ms at 24 VDC	10 ms at 24 VDC
Internal fuse, not user replaceable		3 A, 250 V, slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow

Table A- 30 Sensor power

Technical data	CPU 1212C AC/DC/Relay	CPU 1212C DC/DC/Relay	CPU 1212C DC/DC/DC
Voltage range	20.4 to 28.8 VDC	L+ minus 4 VDC min.	L+ minus 4 VDC min.
Output current rating (max.)	300 mA (short circuit protected)	300 mA (short circuit protected)	300 mA (short circuit protected)

Technical data	CPU 1212C AC/DC/Relay	CPU 1212C DC/DC/Relay	CPU 1212C DC/DC/DC
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	Same as input line	Same as input line
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated

A.3.2 Digital inputs and outputs

Table A- 31 Digital inputs

Technical data	CPU 1212C AC/DC/Relay, DC/DC/Relay, and DC/DC/DC
Number of inputs	8
Type	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.
Logic 1 signal (min.)	15 VDC at 2.5 mA
Logic 0 signal (max.)	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute
Isolation groups	1
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 VDC)	Single phase: 100 KHz (Ia.0 to Ia.5) and 30 KHz (Ia.6 to Ia.7) Quadrature phase: 80 KHz (Ia.0 to Ia.5) and 20 KHz (Ia.6 to Ia.7)
Number of inputs on simultaneously	8
Cable length (meters)	500 m shielded, 300 m unshielded, 50 m shielded for HSC inputs

Table A- 32 Digital outputs

Technical data	CPU 1212C AC/DC/Relay and DC/DC/Relay	CPU 1212C DC/DC/DC
Number of outputs	6	6
Type	Relay, dry contact	Solid state - MOSFET (sourcing)
Voltage range	5 to 30 VDC or 5 to 250 VAC	20.4 to 28.8 VDC
Logic 1 signal at max. current	--	20 VDC min.
Logic 0 signal with 10 KΩ load	--	0.1 VDC max.
Current (max.)	2.0 A	0.5 A
Lamp load	30 W DC / 200 W AC	5 W
ON state resistance	0.2 Ω max. when new	0.6 Ω max.
Leakage current per point	--	10 µA max.
Surge current	7 A with contacts closed	8 A for 100 ms max.
Overload protection	No	No

Technical data	CPU 1212C AC/DC/Relay and DC/DC/Relay	CPU 1212C DC/DC/DC
Isolation (field side to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)	500 VAC for 1 minute
Isolation resistance	100 MΩ min. when new	--
Isolation between open contacts	750 VAC for 1 minute	--
Isolation groups	2	1
Inductive clamp voltage	--	L+ minus 48 VDC, 1 W dissipation
Switching delay (Qa.0 to Qa.3)	10 ms max.	1.0 µs max., off to on 3.0 µs max., on to off
Switching delay (Qa.4 to Qa.5)	10 ms max.	50 µs max., off to on 200 µs max., on to off
Maximum relay switching frequency	1 Hz	--
Pulse Train Output rate (Qa.0 and Qa.2)	Not recommended [1]	100 KHz max., 2 Hz min.[2]
Lifetime mechanical (no load)	10,000,000 open/close cycles	--
Lifetime contacts at rated load	100,000 open/close cycles	--
Behavior on RUN to STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of Outputs On simultaneously	6	6
Cable length (meters)	500 m shielded, 150 m unshielded	500 m shielded, 150 m unshielded

[1] For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

[2] Depending on your pulse receiver and cable, an additional load resistor (at least 10% of rated current) may improve pulse signal quality and noise immunity.

A.3.3 Analog inputs

Table A- 33 Analog inputs

Technical data	Description
Number of inputs	2
Type	Voltage (single-ended)
Range	0 to 10 V
Full-scale range (data word)	0 to 27648
Overshoot range (data word)	27,649 to 32,511 See the table for measurement ranges of the analog inputs for voltage (Page 678).
Overflow (data word)	32,512 to 32,767
Resolution	10 bits
Maximum withstand voltage	35 VDC

Technical data	Description
Smoothing	None, Weak, Medium, or Strong See the table for step response (ms) for the analog inputs of the CPU (Page 677).
Noise rejection	10, 50, or 60 Hz
Impedance	≥100 KΩ
Isolation (field side to logic)	None
Accuracy (25°C / 0 to 55°C)	3.0% / 3.5% of full-scale
Cable length (meters)	100 m, shielded twisted pair

A.3.3.1 Step response of the built-in analog inputs of the CPU

Table A- 34 Step Response (ms), 0V to 10V measured at 95%

Smoothing selection (sample averaging)	Rejection frequency (Integration time)		
	60 Hz	50 Hz	10 Hz
None (1 cycle): No averaging	63 ms	65 ms	130 ms
Weak (4 cycles): 4 samples	84 ms	93 ms	340 ms
Medium (16 cycles): 16 samples	221 ms	258 ms	1210 ms
Strong (32 cycles): 32 samples	424 ms	499 ms	2410 ms
Sample time	**4.17 ms**	**5 ms**	**25 ms**

A.3.3.2 Sample time for the built-in analog ports of the CPU

Table A- 35 Sample time for built-in analog inputs of the CPU

Rejection frequency(Integration time selection)	Sample time
60 Hz(16.6 ms)	4.17 ms
50 Hz (20 ms)	5 ms
10 Hz (100 ms)	25 ms

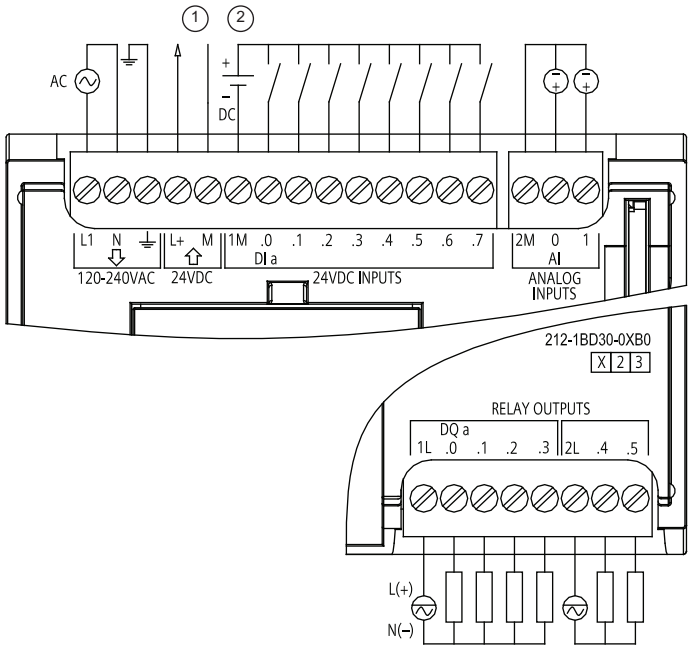
A.3.3.3 Measurement ranges of the analog inputs for voltage

Table A- 36 Analog input representation for voltage

System		Voltage Measuring Range					
Decimal	Hexadecimal	±10 V	±5 V	±2.5 V		0 to 10 V	
32767	7FFF	11.851 V	5.926 V	2.963 V	Overflow	11.851 V	Overflow
32512	7F00						
32511	7EFF	11.759 V	5.879 V	2.940 V	Overshoot range	11.759 V	Overshoot range
27649	6C01						
27648	6C00	10 V	5 V	2.5 V	Rated range	10 V	Rated range
20736	5100	7.5 V	3.75 V	1.875 V		7.5 V	
1	1	361.7 µV	180.8 µV	90.4 µV		361.7 µV	
0	0	0 V	0 V	0 V		0 V	
-1	FFFF					Negative values are not supported	
-20736	AF00	-7.5 V	-3.75 V	-1.875 V			
-27648	9400	-10 V	-5 V	-2.5 V			
-27649	93FF				Undershoot range		
-32512	8100	-11.759 V	-5.879 V	-2.940 V			
-32513	80FF				Underflow		
-32768	8000	-11.851 V	-5.926 V	-2.963 V			

A.3.4 Wiring diagrams

Table A- 37 CPU 1212C AC/DC/Relay (6ES7 212-1BD30-0XB0)



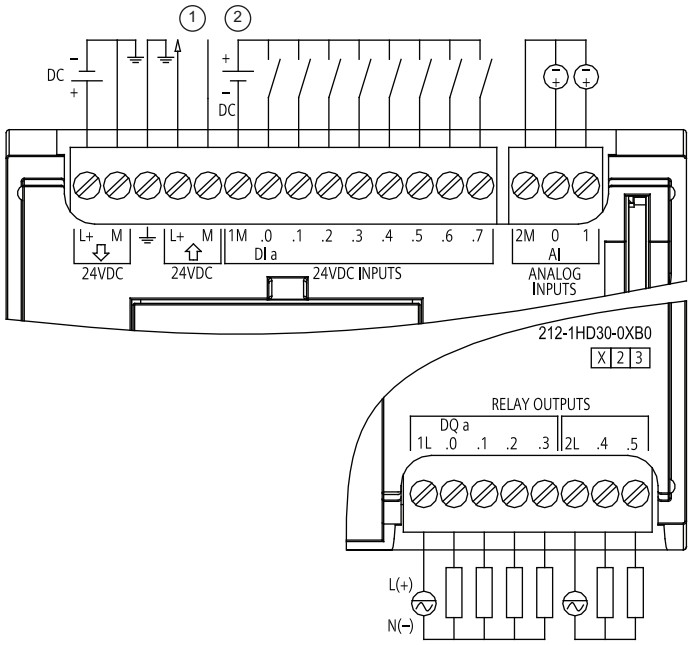
① 24 VDC Sensor Power Out

For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).

For sourcing inputs, connect "+" to "M".

Table A- 38 CPU 1212C DC/DC/Relay (6ES7 212-1HD30-0XB0)



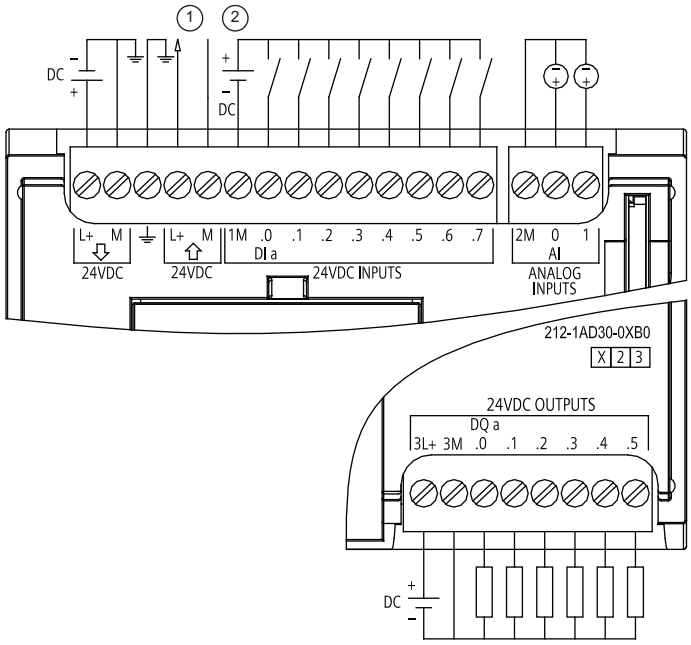
① 24 VDC Sensor Power Out

For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).

For sourcing inputs, connect "+" to "M".

Table A- 39 CPU 1212C DC/DC/DC (6ES7-212-1AD30-0XB0)



① 24 VDC Sensor Power Out

For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).

For sourcing inputs, connect "+" to "M".

Note

Unused analog inputs should be shorted.

A.4 CPU 1214C

A.4.1 General specifications and features

Table A- 40 General

Technical data	CPU 1214C AC/DC/Relay	CPU 1214C DC/DC/Relay	CPU 1214C DC/DC/DC
Order number	6ES7 214-1BE30-0XB0	6ES7 214-1HE30-0XB0	6ES7 214-1AE30-0XB0
Dimensions W x H x D (mm)	110 x 100 x 75	110 x 100 x 75	110 x 100 x 75
Weight	475 grams	435 grams	415 grams
Power dissipation	14 W	12 W	12 W
Current available (SM and CM bus)	1600 mA max. (5 VDC)	1600 mA max. (5 VDC)	1600 mA max. (5 VDC)

Technical data	CPU 1214C AC/DC/Relay	CPU 1214C DC/DC/Relay	CPU 1214C DC/DC/DC
Current available (24 VDC)	400 mA max. (sensor power)	400 mA max. (sensor power)	400 mA max. (sensor power)
Digital input current consumption (24VDC)	4 mA/input used	4 mA/input used	4 mA/input used

Table A- 41 CPU features

Technical data		Description
User memory[1]	Work	50 Kbytes
	Load	2 Mbytes
	Retentive	2 Kbytes
On-board digital I/O		14 inputs/10 outputs
On-board analog I/O		2 inputs
Process image size		1024 bytes of inputs (I)/1024 bytes of outputs (Q)
Bit memory (M)		8192 bytes
Temporary (local) memory		• 16 Kbytes for startup and program cycle (including associated FBs and FCs) • 4 Kbytes for standard interrupt events including FBs and FCs • 4 Kbytes for error interrupt events including FBs and FCs
Signal modules expansion		8 SMs max.
Signal board expansion		1 SB max.
Communication module expansion		3 CMs max.
High-speed counters		6 total • Single phase: 3 at 100 kHz and 3 at 30 kHz clock rate • Quadrature phase: 3 at 80 kHz and 3 at 20 kHz clock rate
Pulse outputs [2]		2
Pulse catch inputs		14
Time delay / cyclic interrupts		4 total with 1 ms resolution
Edge interrupts		12 rising and 12 falling (14 and 14 with optional signal board)
Memory card		SIMATIC Memory Card (optional)
Real time clock accuracy		+/- 60 seconds/month
Real time clock retention time		10 days typ./6 days min. at 40°C (maintenance-free Super Capacitor)

1 The size of the user program, data, and configuration is limited by the available load memory and work memory in the CPU. There is no specific limit to the number of OB, FC, FB and DB blocks supported or to the size of a particular block; the only limit is due to overall memory size.

2 For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

Table A- 42 Performance

Type of instruction	Execution speed
Boolean	0.1 µs/instruction
Move Word	12 µs/instruction
Real math	18 µs/instruction

Table A- 43 Blocks, timers and counters supported by S7-1200

Element		Description
Blocks	Type	OB, FB, FC, DB
	Size	25 Kbytes (CPU 1211C and CPU 1212C) 50 Kbytes (CPU 1214C)
	Quantity	Up to 1024 blocks total (OBs + FBs + FCs + DBs)
	Address range for FBs, FCs, and DBs	1 to 65535 (such as FB 1 to FB 65535)
	Nesting depth	16 from the program cycle or start up OB; 4 from the time delay interrupt, time-of-day interrupt, cyclic interrupt, hardware interrupt, time error interrupt, or diagnostic error interrupt OB
	Monitoring	Status of 2 code blocks can be monitored simultaneously
OBs	Program cycle	Multiple: OB 1, OB 200 to OB 65535
	Startup	Multiple: OB 100, OB 200 to OB 65535
	Time-delay interrupts and cyclic interrupts	4[1] (1 per event): OB 200 to OB 65535
	Hardware interrupts (edges and HSC)	50 (1 per event): OB 200 to OB 65535
	Time error interrupts	1: OB 80
	Diagnostic error interrupts	1: OB 82
Timers	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, 16 bytes per timer
Counters	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, size dependent upon count type • SInt, USInt: 3 bytes • Int, UInt: 6 bytes • DInt, UDInt: 12 bytes

[1] Time-delay and cyclic interrupts use the same resources in the CPU. You can have only a total of 4 of these interrupts (time-delay plus cyclic interrupts). You cannot have 4 time-delay interrupts and 4 cyclic interrupts.

Table A- 44 Communication

Technical data	Description
Number of ports	1
Type	Ethernet
HMI device[1]	3
Programming device (PG)	1
Connections	• 8 for Open User Communication (active or passive): TSEND_C, TRCV_C, TCON, TDISCON, TSEND, and TRCV • 3 for server GET/PUT (CPU-to-CPU) S7 communication • 8 for client GET/PUT (CPU-to-CPU) S7 communication
Data rates	10/100 Mb/s
Isolation (external signal to PLC logic)	Transformer isolated, 1500 VDC
Cable type	CAT5e shielded

1 The CPU provides dedicated HMI connections to support up to 3 HMI devices. (You can have up to 2 SIMATIC Comfort panels.) The total number of HMI is affected by the types of HMI panels in your configuration. For example, you could have up to three SIMATIC Basic panels connected to your CPU, or you could have up to two SIMATIC Comfort panels with one additional Basic panel.

Table A- 45 Power supply

Technical data		CPU 1214C AC/DC/Relay	CPU 1214C DC/DC/Relay	CPU 1214C DC/DC/DC
Voltage range		85 to 264 VAC	20.4 to 28.8 VDC	
Line frequency		47 to 63 Hz	--	
Input current (max. load)	CPU only	100 mA at 120 VAC 50 mA at 240 VAC	500 mA at 24 VDC	
	CPU with all expansion accessories	300 mA at 120 VAC 150 mA at 240 VAC	1500 mA at 24 VDC	
Inrush current (max.)		20 A at 264 VAC	12 A at 28.8 VDC	
Isolation (input power to logic)		1500 VAC	Not isolated	
Ground leakage, AC line to functional earth		0.5 mA max.	-	
Hold up time (loss of power)		20 ms at 120 VAC 80 ms at 240 VAC	10 ms at 24 VDC	
Internal fuse, not user replaceable		3 A, 250 V, slow blow		

Table A- 46 Sensor power

Technical data	CPU 1214C AC/DC/Relay	CPU 1214C DC/DC/Relay	CPU 1214C DC/DC/DC
Voltage range	20.4 to 28.8 VDC	L+ minus 4 VDC min.	
Output current rating (max.)	400 mA (short circuit protected)		

Technical data	CPU 1214C AC/DC/Relay	CPU 1214C DC/DC/Relay	CPU 1214C DC/DC/DC
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	Same as input line	
Isolation (CPU logic to sensor power)	Not isolated		

A.4.2 Digital inputs and outputs

Table A- 47 Digital inputs

Technical data	CPU 1214C AC/DC/Relay	CPU 1214C DC/DC/Relay	CPU 1214C DC/DC/DC
Number of inputs	14		
Type	Sink/Source (IEC Type 1 sink)		
Rated voltage	24 VDC at 4 mA, nominal		
Continuous permissible voltage	30 VDC, max.		
Surge voltage	35 VDC for 0.5 sec.		
Logic 1 signal (min.)	15 VDC at 2.5 mA		
Logic 0 signal (max.)	5 VDC at 1 mA		
Isolation (field side to logic)	500 VAC for 1 minute		
Isolation groups	1		
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)		
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 VDC)	Single phase: 100 KHz (Ia.0 to Ia.5) and 30 KHz (Ia.6 to Ib.5) Quadrature phase: 80 KHz (Ia.0 to Ia.5) and 20 KHz (Ia.6 to Ib.5)		
Number of inputs on simultaneously	14		
Cable length (meters)	500 m shielded, 300 m unshielded, 50 m shielded for HSC inputs		

Table A- 48 Digital outputs

Technical data	CPU 1214C AC/DC/Relay and DC/DC/Relay	CPU 1214C DC/DC/DC
Number of outputs	10	10
Type	Relay, dry contact	Solid state - MOSFET (sourcing)
Voltage range	5 to 30 VDC or 5 to 250 VAC	20.4 to 28.8 VDC
Logic 1 signal at max. current	--	20 VDC min.
Logic 0 signal with 10 KΩ load	--	0.1 VDC max.
Current (max.)	2.0 A	0.5 A
Lamp load	30 W DC / 200 W AC	5 W
ON state resistance	0.2 Ω max. when new	0.6 Ω max.
Leakage current per point	--	10 µA max.
Surge current	7 A with contacts closed	8 A for 100 ms max.
Overload protection	No	No

Technical data	CPU 1214C AC/DC/Relay and DC/DC/Relay	CPU 1214C DC/DC/DC
Isolation (field side to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)	500 VAC for 1 minute
Isolation resistance	100 MΩ min. when new	--
Isolation between open contacts	750 VAC for 1 minute	--
Isolation groups	2	1
Inductive clamp voltage	--	L+ minus 48 VDC, 1 W dissipation
Switching delay (Qa.0 to Qa.3)	10 ms max.	1.0 µs max., off to on 3.0 µs max., on to off
Switching delay (Qa.4 to Qb.1)	10 ms max.	50 µs max., off to on 200 µs max., on to off
Maximum relay switching frequency	1 Hz	--
Pulse Train Output rate (Qa.0 and Qa.2)	Not recommended [1]	100 KHz max., 2 Hz min. [2]
Lifetime mechanical (no load)	10,000,000 open/close cycles	--
Lifetime contacts at rated load	100,000 open/close cycles	--
Behavior on RUN to STOP	Last value or substitute value (default value 0)	
Number of Outputs On simultaneously	10	
Cable length (meters)	500 m shielded, 150 m unshielded	

1 For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

2 Depending on your pulse receiver and cable, an additional load resistor (at least 10% of rated current) may improve pulse signal quality and noise immunity.

A.4.3 Analog inputs

Table A- 49 Analog inputs

Technical data	Description
Number of inputs	2
Type	Voltage (single-ended)
Range	0 to 10 V
Full-scale range (data word)	0 to 27648
Overshoot range (data word)	27,649 to 32,511 See the table for measurement ranges of the analog inputs for voltage (Page 687).
Overflow (data word)	32,512 to 32,767
Resolution	10 bits
Maximum withstand voltage	35 VDC

Technical data	Description
Smoothing	None, Weak, Medium, or Strong See the table for step response (ms) for the analog inputs of the CPU (Page 686).
Noise rejection	10, 50, or 60 Hz
Impedance	≥100 KΩ
Isolation (field side to logic)	None
Accuracy (25°C / 0 to 55°C)	3.0% / 3.5% of full-scale
Cable length (meters)	100 m, shielded twisted pair

A.4.3.1 Step response of the built-in analog inputs of the CPU

Table A- 50 Step Response (ms), 0V to 10V measured at 95%

Smoothing selection (sample averaging)	Rejection frequency (Integration time)		
	60 Hz	50 Hz	10 Hz
None (1 cycle): No averaging	63 ms	65 ms	130 ms
Weak (4 cycles): 4 samples	84 ms	93 ms	340 ms
Medium (16 cycles): 16 samples	221 ms	258 ms	1210 ms
Strong (32 cycles): 32 samples	424 ms	499 ms	2410 ms
Sample time	**4.17 ms**	**5 ms**	**25 ms**

A.4.3.2 Sample time for the built-in analog ports of the CPU

Table A- 51 Sample time for built-in analog inputs of the CPU

Rejection frequency(Integration time selection)	Sample time
60 Hz(16.6 ms)	4.17 ms
50 Hz (20 ms)	5 ms
10 Hz (100 ms)	25 ms

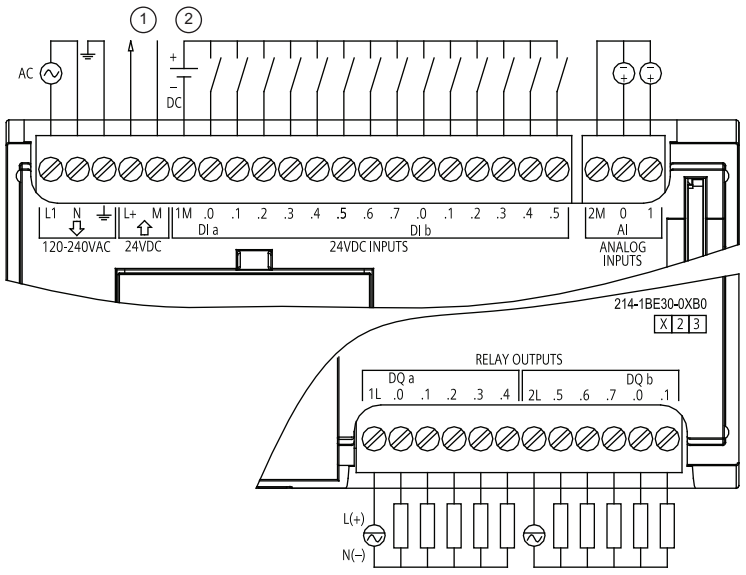
A.4.3.3 Measurement ranges of the analog inputs for voltage

Table A- 52 Analog input representation for voltage

System		Voltage Measuring Range					
Decimal	Hexadecimal	±10 V	±5 V	±2.5 V		0 to 10 V	
32767	7FFF	11.851 V	5.926 V	2.963 V	Overflow	11.851 V	Overflow
32512	7F00						
32511	7EFF	11.759 V	5.879 V	2.940 V	Overshoot range	11.759 V	Overshoot range
27649	6C01						
27648	6C00	10 V	5 V	2.5 V	Rated range	10 V	Rated range
20736	5100	7.5 V	3.75 V	1.875 V		7.5 V	
1	1	361.7 µV	180.8 µV	90.4 µV		361.7 µV	
0	0	0 V	0 V	0 V		0 V	
-1	FFFF					Negative values are not supported	
-20736	AF00	-7.5 V	-3.75 V	-1.875 V			
-27648	9400	-10 V	-5 V	-2.5 V			
-27649	93FF				Undershoot range		
-32512	8100	-11.759 V	-5.879 V	-2.940 V			
-32513	80FF				Underflow		
-32768	8000	-11.851 V	-5.926 V	-2.963 V			

A.4.4 CPU 1214C Wiring Diagrams

Table A- 53 CPU 1214C AC/DC/Relay (6ES7 214-1BE30-0XB0)



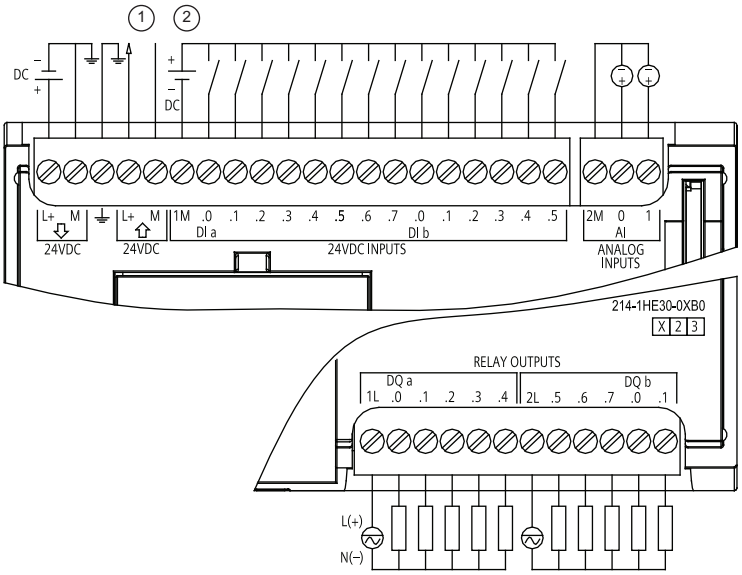
① 24 VDC Sensor Power Out

For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).

For sourcing inputs, connect "+" to "M".

Table A- 54 CPU 1214C DC/DC/Relay (6ES7 214-1HE30-0XB0)



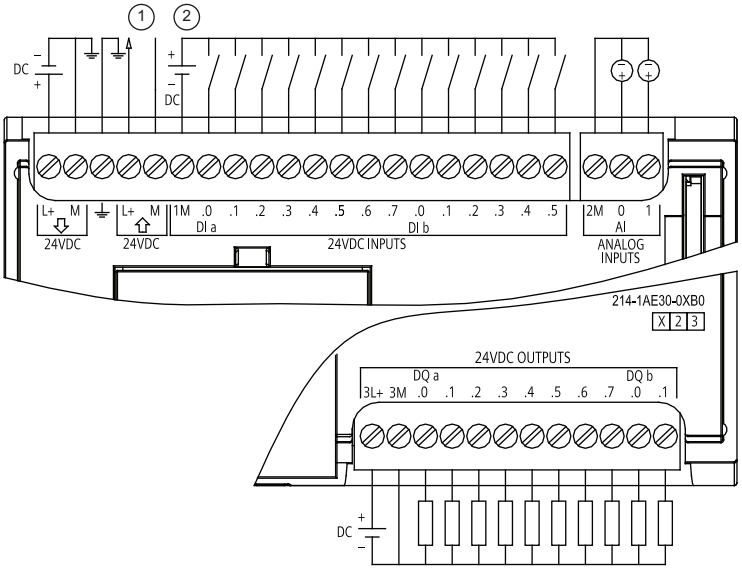
① 24 VDC Sensor Power Out

For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).

For sourcing inputs, connect "+" to "M".

Table A- 55 CPU 1214C DC/DC/DC (6ES7 214-1AE30-0XB0)



① 24 VDC Sensor Power Out

For additional noise immunity, connect "M" to chassis ground even if not using sensor supply.

② For sinking inputs, connect "-" to "M" (shown).

For sourcing inputs, connect "+" to "M".

Note

Unused analog inputs should be shorted.

A.5 Digital signal modules (SMs)

A.5.1 SM 1221 Digital Input Specifications

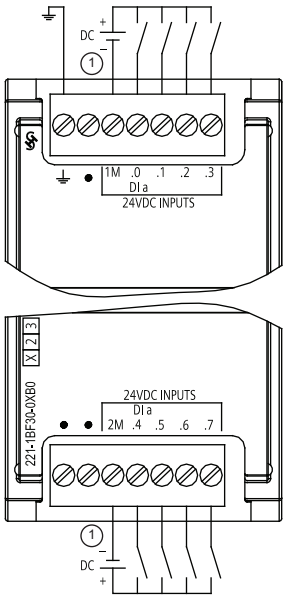
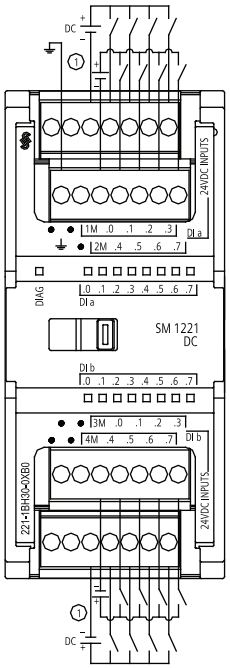
Table A- 56 General specifications

Model	SM 1221 DI 8 x 24 VDC	SM 1221 DI 16 x 24 VDC
Order number	6ES7 221-1BF30-0XB0	6ES7 221-1BH30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	45 x 100 x 75
Weight	170 grams	210 grams
Power dissipation	1.5 W	2.5 W
Current consumption (SM Bus)	105 mA	130 mA
Current consumption (24 VDC)	4 mA / input used	4 mA / input used

Table A- 57 Digital inputs

Model	SM 1221 DI 8 x 24 VDC	SM 1221 DI 16 x 24 VDC
Number of inputs	8	16
Type	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.
Logic 1 signal (min.)	15 VDC at 2.5 mA	15 VDC at 2.5 mA
Logic 0 signal (max.)	5 VDC at 1 mA	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	2	4
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)
Number of inputs on simultaneously	8	16
Cable length (meters)	500 m shielded, 300 unshielded	500 m shielded, 300 unshielded

Table A- 58 Wiring diagrams for the digital input SMs

SM 1221 DI 8 x 24 VDC (6ES7 221-1BF30-0XB0)	SM 1221 DI 16 x 24 VDC (6ES7 221-1BH30-0XB0)
	

① For sinking inputs, connect "-" to "M" (shown). For sourcing inputs, connect "+" to "M".

A.5.2 SM 1222 8-Point Digital Output Specifications

Table A- 59 General specifications

Model	SM 1222 DQ 8 x Relay	SM 1222 DQ8 RLY Changeover	SM 1222 DQ 8 x 24 VDC
Order number	6ES7 222-1HF30-0XB0	6ES7 222-1XF30-0XB0	6ES7 222-1BF30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	70 x 100 x 75	45 x 100 x 75
Weight	190 grams	310 grams	180 grams
Power dissipation	4.5 W	5 W	1.5 W
Current consumption (SM Bus)	120 mA	140 mA	120 mA
Current consumption (24 VDC)	11 mA / Relay coil used	16.7 mA/Relay coil used	11 mA / Relay coil used

Table A- 60 Digital outputs

Model	SM 1222 DQ 8 x Relay	SM 1222 DQ8 RLY Changeover	SM 1222 DQ 8 x 24 VDC
Number of outputs	8	8	8
Type	Relay, dry contact	Relay change over contact	Solid state - MOSFET (sourcing)
Voltage range	5 to 30 VDC or 5 to 250 VAC	5 to 30 VDC or 5 to 250 VAC	20.4 to 28.8 VDC
Logic 1 signal at max. current	--	--	20 VDC min.
Logic 0 signal with 10K Ω load	--	--	0.1 VDC max
Current (max.)	2.0 A	2.0 A	0.5 A
Lamp load	30 W DC/200 W AC	30 W DC/200 W AC	5W
ON state contact resistance	0.2 Ω max. when new	0.2 Ω max. when new	0.6 Ω max.
Leakage current per point	--	--	10 µA max.
Surge current	7 A with contacts closed	7 A with contacts closed	8 A for 100 ms max.
Overload protection	No	No	No
Isolation (field side to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)	1500 VAC for 1 minute (coil to contact)	500 VAC for 1 minute
Isolation resistance	100 MΩ min. when new	100 MΩ min. when new	--
Isolation between open contacts	750 VAC for 1 minute	750 VAC for 1 minute	--
Isolation groups	2	8	1
Current per common (max.)	10 A	2 A	4 A
Inductive clamp voltage	--	--	L+ minus 48 V, 1 W dissipation
Switching delay	10 ms max.	10 ms max	50 µs max. off to on 200 µs max. on to off
Maximum relay switching frequency	1 Hz	1 Hz	--
Lifetime mechanical (no load)	10,000,000 open/close cycles	10,000,000 open/close cycles	--
Lifetime contacts at rated load	100,000 open/close cycles	100,000 open/close cycles	--
Behavior on RUN to STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8	8	8
Cable length (meters)	500 m shielded, 150 m unshielded	500 m shielded, 150 m unshielded	500 m shielded, 150 m unshielded

A.5.3 SM 1222 16-Point Digital Output Specifications

Table A- 61 General specifications

Model	SM 1222 DQ 16 x Relay	SM 1222 DQ 16 x 24 VDC
Order number	6ES7 222-1HH30-0XB0	6ES7 222-1BH30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	45 x 100 x 75
Weight	260 grams	220 grams
Power dissipation	8.5 W	2.5 W
Current consumption (SM Bus)	135 mA	140 mA
Current consumption (24 VDC)	11 mA / Relay coil used	-

Table A- 62 Digital outputs

Model	SM1222 DQ 16 x Relay	SM1222 DQ 16 x 24 VDC
Number of outputs	16	16
Type	Relay, dry contact	Solid state - MOSFET (sourcing)
Voltage range	5 to 30 VDC or 5 to 250 VAC	20.4 to 28.8 VDC
Logic 1 signal at max. current	-	20 VDC min.
Logic 0 signal with 10K Ω load	-	0.1 VDC max.
Current (max.)	2.0 A	0.5 A
Lamp load	30 W DC/200 W AC	5W
ON state contact resistance	0.2 Ω max. when new	0.6 Ω max.
Leakage current per point	--	10 µA max.
Surge current	7 A with contacts closed	8 A for 100 ms max.
Overload protection	No	No
Isolation (field side to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)	500 VAC for 1 minute
Isolation resistance	100 MΩ min. when new	-
Isolation between open contacts	750 VAC for 1 minute	-
Isolation groups	4	1
Current per common (max.)	10 A	8 A
Inductive clamp voltage	-	L+ minus 48 V, 1 W dissipation
Switching delay	10 ms max.	50 µs max. off to on 200 µs max. on to off
Maximum relay switching frequency	1 Hz	-
Lifetime mechanical (no load)	10,000,000 open/close cycles	-
Lifetime contacts at rated load	100,000 open/close cycles	-
Behavior on RUN to STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)

Model	SM1222 DQ 16 x Relay	SM1222 DQ 16 x 24 VDC
Number of outputs on simultaneously	16	16
Cable length (meters)	500 m shielded, 150 m unshielded	500 m shielded, 150 m unshielded

Table A- 63 Wiring diagrams for the 8-point digital output SMs

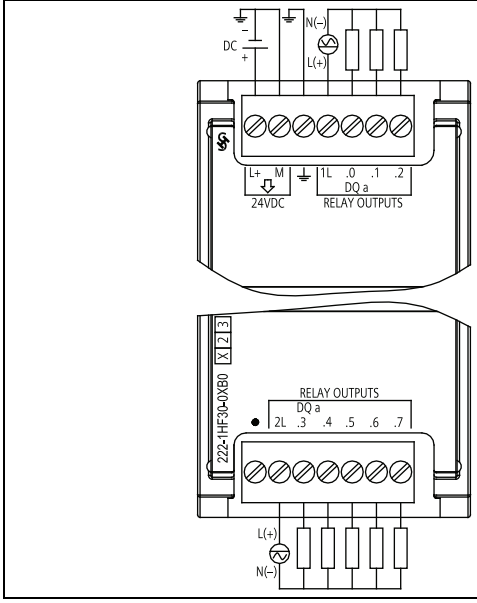
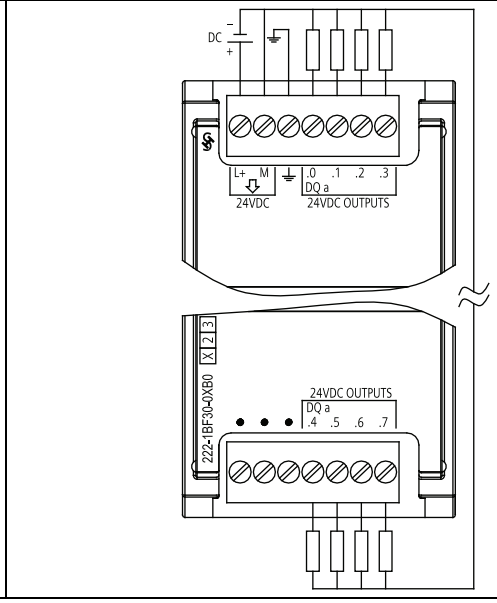
SM 1222 DQ 8 x Relay (6ES7 222-1HF30-0XB0)	SM 1222 DQ 8 x 24 VDC (6ES7 222-1BF30-0XB0)
	

Table A- 64 Wiring diagram for the 8-point digital output relay changeover SM

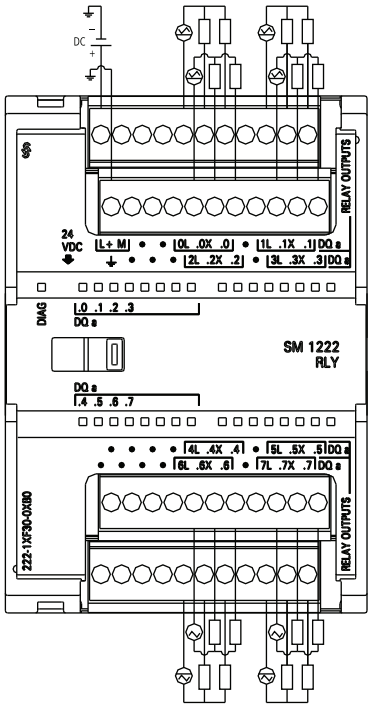
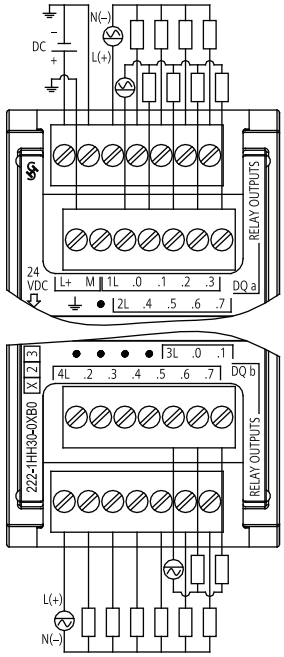
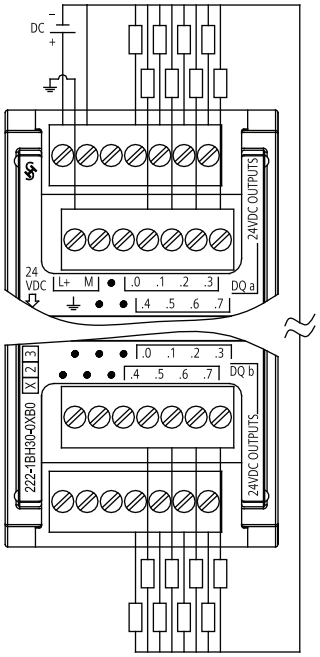
SM 1222 DQ 8 x Relay Changeover (6ES7 222-1XF30-0XB0)	
	

Table A- 65 Wiring diagrams for the 16-point digital output SMs

SM 1222 DQ 16 x Relay (6ES7 222-1HH30-0XB0)	SM 1222 DQ 16 x 24 VDC (6ES7 222-1BH30-0XB0)
	

A.5.4 SM 1223 Digital Input/Output VDC Specifications

Table A- 66 General specifications

Model	SM 1223 DI 8 x 24 VDC, DQ 8 x Relay	SM 1223 DI 16 x 24 VDC, DQ 16 x Relay	SM 1223 DI 8 x 24 VDC, DQ 8 x 24 VDC	SM 1223 DI 16 x 24 VDC, DQ 16 x 24 VDC
Order number	6ES7 223-1PH30-0XB0	6ES7 223-1PL30-0XB0	6ES7 223-1BH30-0XB0	6ES7 223-1BL30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	70 x 100 x 75	45 x 100 x 75	70 x 100 x 75
Weight	230 grams	350 grams	210 grams	310 grams
Power dissipation	5.5 W	10 W	2.5 W	4.5 W
Current consumption (SM Bus)	145 mA	180 mA	145 mA	185 mA
Current consumption (24 VDC)	4 mA / Input used 11 mA / Relay coil used		4 mA / Input used	

Table A- 67 Digital inputs

Model	SM 1223 DI 8 x 24 VDC, DQ 8 x Relay	SM 1223 DI 16 x 24 VDC, DQ 16 x Relay	SM 1223 DI 8 x 24 VDC, DQ 8 x 24 VDC	SM 1223 DI 16 x 24 VDC, DQ 16 x 24 VDC
Number of inputs	8	16	8	16
Type	Sink/Source (IEC Type 1 sink)			
Rated voltage	24 VDC at 4 mA, nominal			
Continuous permissible voltage	30 VDC max.			
Surge voltage	35 VDC for 0.5 sec.			
Logic 1 signal (min.)	15 VDC at 2.5 mA			
Logic 0 signal (max.)	5 VDC at 1 mA			
Isolation (field side to logic)	500 VAC for 1 minute			
Isolation groups	2	2	2	2
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4			
Number of inputs on simultaneously	8	16	8	16
Cable length (meters)	500 m shielded, 300 m unshielded			

Table A- 68 Digital outputs

Model	SM 1223 DI 8 x 24 VDC, DQ 8 x Relay	SM 1223 DI 16 x 24 VDC, DQ 16 x Relay	SM 1223 DI 8 x 24 VDC, DQ 8 x 24 VDC	SM 1223 DI 16 x 24 VDC, DQ 16 x 24 VDC
Number of outputs	8	16	8	16
Type	Relay, dry contact		Solid state - MOSFET (sourcing)	
Voltage range	5 to 30 VDC or 5 to 250 VAC		20.4 to 28.8 VDC	
Logic 1 signal at max. current	--		20 VDC, min.	

Model	SM 1223 DI 8 x 24 VDC, DQ 8 x Relay	SM 1223 DI 16 x 24 VDC, DQ 16 x Relay	SM 1223 DI 8 x 24 VDC, DQ 8 x 24 VDC	SM 1223 DI 16 x 24 VDC, DQ 16 x 24 VDC
Logic 0 signal with 10 KΩ load	--		0.1 VDC, max.	
Current (max.)	2.0 A		0.5 A	
Lamp load	30 W DC / 200 W AC		5 W	
ON state contact resistance	0.2 Ω max. when new		0.6 Ω max.	
Leakage current per point	--		10 µA max.	
Surge current	7 A with contacts closed		8 A for 100 ms max.	
Overload protection	No			
Isolation (field side to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)		500 VAC for 1 minute	
Isolation resistance	100 MΩ min. when new		--	
Isolation between open contacts	750 VAC for 1 minute		--	
Isolation groups	2	4	1	1
Current per common	10A	8 A	4 A	8 A
Inductive clamp voltage	--		L+ minus 48 V, 1 W dissipation	
Switching delay	10 ms max.		50 µs max. off to on 200 µs max. on to off	
Maximum relay switching frequency	1 Hz		--	
Lifetime mechanical (no load)	10,000,000 open/close cycles		--	
Lifetime contacts at rated load	100,000 open/close cycles		--	
Behavior on RUN to STOP	Last value or substitute value (default value 0)			
Number of outputs on simultaneously	8	16	8	16
Cable length (meters)	500 m shielded, 150 m unshielded			

Table A- 69 Wiring diagrams for the digital input VDC/output relay SMs

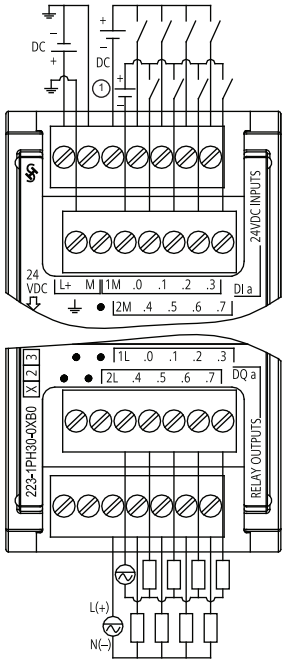
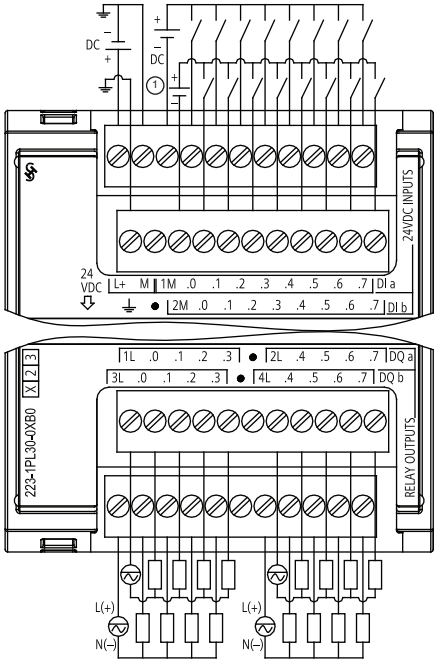
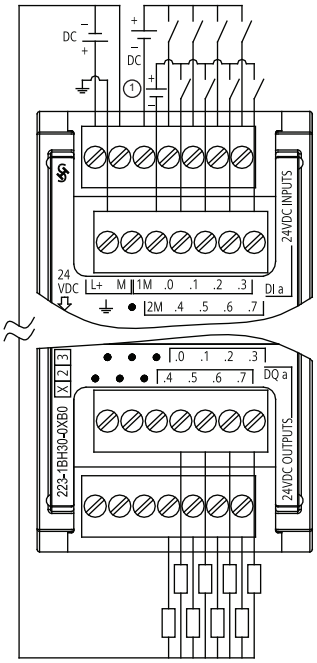
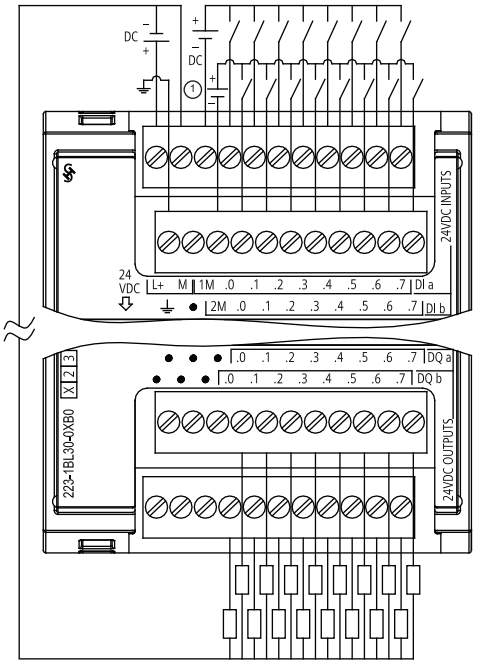
SM 1223 DI 8 x 24 VDC, DQ 8 x Relay (6ES7 223-1PH30-0XB0)	SM 1223 DI 16 x 24 VDC, DQ 16 x Relay (6ES7 223-1PL30-0XB0)	Notes
		① For sinking inputs, connect "-" to "M" (shown). For sourcing inputs, connect "+" to "M".

Table A- 70 Wiring diagrams for the digital input VDC/output SMs

SM 1223 DI 8 x 24 VDC, DQ 8 x 24 VDC (6ES7 223-1BH30-0XB0)	SM 1223 DI 16 x 24 VDC, DQ 16 x 24 VDC (6ES7 223-1BL30-0XB0)	Notes
		① For sinking inputs, connect "-" to "M" (shown). For sourcing inputs, connect "+" to "M".

A.5.5 SM 1223 Digital Input/Output AC Specifications

Table A- 71 General specifications

Model	SM 1223 DI 8 x120/230 VAC / DQ 8 x Relay
Order number	6ES7 223-1QH30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75 mm
Weight	190 grams
Power dissipation	7.5 W
Current consumption (SM Bus)	120 mA
Current consumption (24 VDC)	11 mA per output when on

Table A- 72 Digital inputs

Model	SM 1223 DI 8 x 120/230 VAC / DQ 8 x Relay
Number of inputs	8
Type	IEC Type 1
Rated voltage	120 VAC at 6 mA, 230 VAC at 9 mA
Continuous permissible voltage	264 VAC
Surge voltage	--
Logic 1 signal (min.)	79 VAC at 2.5 mA
Logic 0 signal (max.)	20 VAC at 1 mA
Leakage current (max.)	1 mA
Isolation (field side to logic)	1500 VAC for 1 minute
Isolation groups[1]	4
Input delay times	Typical: 0.2 to 12.8 ms, user selectable Maximum: -
Connection of 2 wire proximity sensor (Bero) (max.)	1 mA
Cable length	Unshielded: 300 meters Shielded: 500 meters
Number of inputs on simultaneously	8

[1] Channels within a group must be of the same phase.

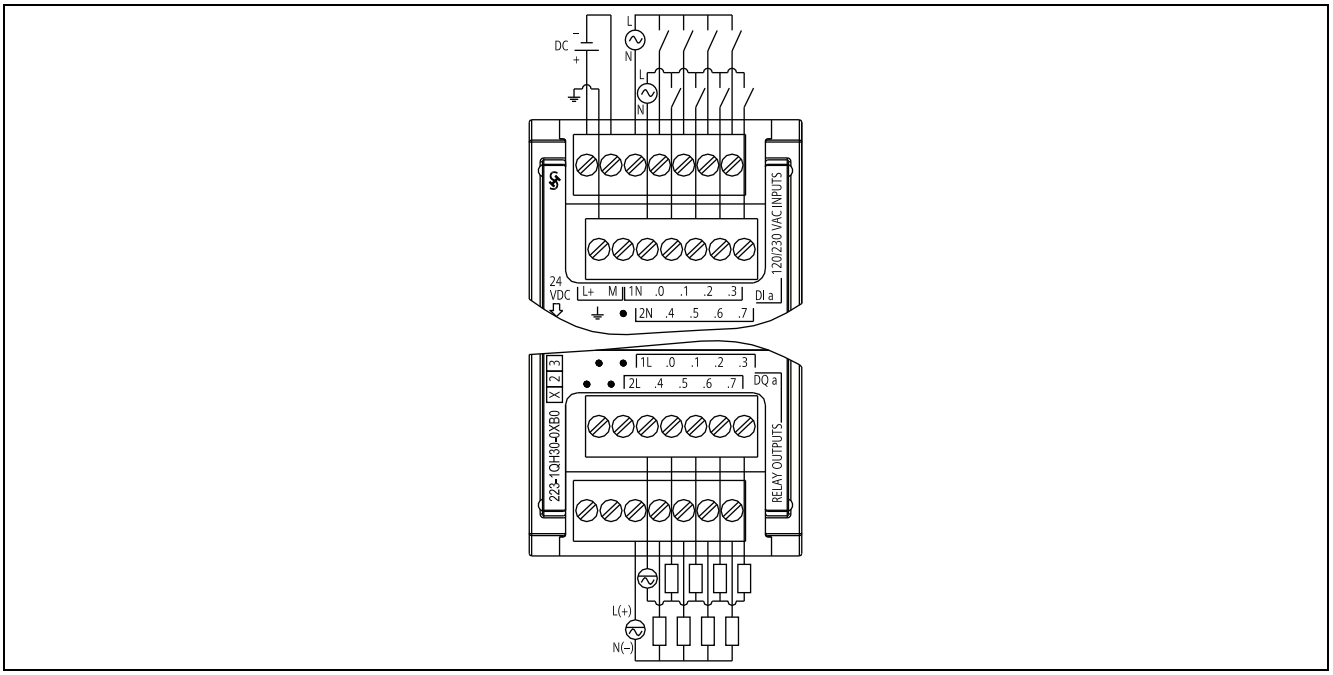
Table A- 73 Digital outputs

Model	SM 1223 DI 8 x 120/230 VAC / DQ 8 x Relay
Number of outputs	8
Type	Relay, dry contact
Voltage range	5 to 30 VDC or 5 to 250 VAC
Logic 1 signal at max. current	--
Logic 0 signal with 10K Ω load	--
Current (max.)	2.0 A
Lamp load	30 W DC / 200 W AC
ON state contact resistance	0.2 Ω max. when new
Leakage current per point	--
Surge current	7 A with contacts closed
Overload protection	No
Isolation (field side to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	100 MΩ min. when new
Isolation between open contacts	750 VAC for 1 minute
Isolation groups	2
Current per common (max.)	10 A
Inductive clamp voltage	--
Switching delay (max.)	10 ms
Maximum relay switching frequency	1 Hz
Lifetime mechanical (no load)	10,000,000 open/close cycles
Lifetime contacts at rated load	1000,000 open/close cycles
Behavior on RUN to STOP	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8
Cable length (meters)	500 m shielded, 150 m unshielded

Note

The SM 1223 DI 8 x 120/230 VAC, DQ 8 x Relay (6ES7 223-1QH30-0XB0) signal module is approved for use in Class 1, Division 2, Gas Group A, B, C, D, Temperature Class T4 Ta = 40° C.

Table A- 74 SM 1223 DI 8 x 120/230 VAC, DQ 8 x Relay (6ES7 223-1QH30-0XB0)



A.6 Analog signal modules (SMs)

A.6.1 SM 1231 analog input module specifications

Table A- 75 General specifications

Model	SM 1231 AI 4 x 13 bit	SM 1231 AI 8 x 13 bit
Order number	6ES7 231-4HD30-0XB0	6ES7 231-4HF30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	45 x 100 x 75
Weight	180 grams	180 grams
Power dissipation	1.5 W	1.5 W
Current consumption (SM Bus)	80 mA	90 mA
Current consumption (24 VDC)	45 mA	45 mA

Table A- 76 Analog inputs

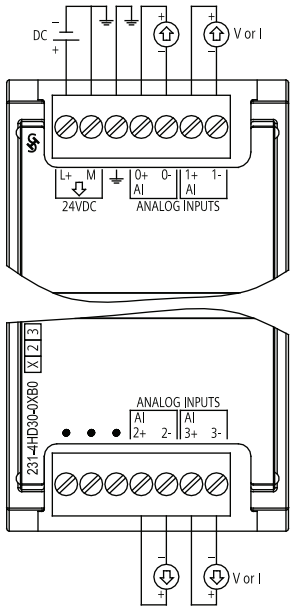
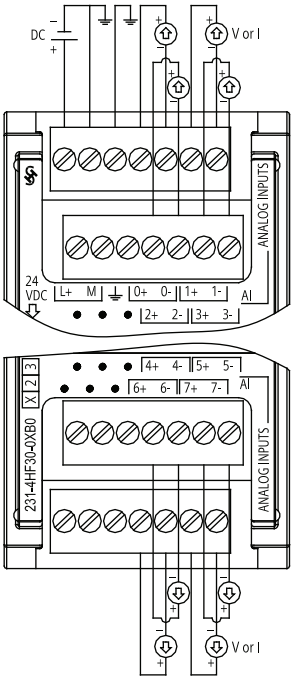
Model	SM 1231 AI 4 x 13 bit	SM 1231 AI 8 x 13 bit
Number of inputs	4	8
Type	Voltage or current (differential): Selectable in groups of 2	
Range	±10 V, ±5 V, ±2.5 V, or 0 to 20 mA	
Full scale range (data word)	-27,648 to 27,648	
Overshoot/undershoot range (data word)	Voltage: 32,511 to 27,649 / -27,649 to -32,512 Current: 32,511 to 27,649 / 0 to -4864 Refer to the section on analog input ranges for voltage and current (Page 708).	
Overflow/underflow (data word)	Voltage: 32,767 to 32,512 / -32,513 to -32,768 Current: 32,767 to 32,512 / -4865 to -32,768 Refer to the section on input ranges for voltage and current (Page 708).	
Resolution	12 bits + sign bit	
Maximum withstand voltage/current	±35 V / ±40 mA	
Smoothing	None, weak, medium, or strong Refer to the section on step response times (Page 707).	
Noise rejection	400, 60, 50, or 10 Hz Refer to the section on sample rates (Page 707).	
Input impedance	≥ 9 MΩ (voltage) / 280 Ω (current)	
Isolation (field side to logic)	None	
Accuracy (25°C / 0 to 55°C)	±0.1% / ±0.2% of full scale	
Measuring principle	Actual value conversion	
Common mode rejection	40 dB, DC to 60 Hz	
Operational signal range	Signal plus common mode voltage must be less than +12 V and greater than -12 V	
Cable length (meters)	100 m, twisted and shielded	

Table A- 77 Diagnostics

Model	SM 1231 AI 4 x 13 bit	SM 1231 AI 8 x 13 bit
Overflow/underflow	Yes [1]	Yes
24 VDC low voltage	Yes	Yes

[1] For SM 1231 AI 4 x 13 bit: If a voltage greater than +30 VDC or less than -15 VDC is applied to the input, the resulting value will be unknown and the corresponding overflow or underflow may not be active.

Table A- 78 Wiring diagrams for the analog input SMs

SM 1231 AI 4 x 13 bit (6ES7 231-4HD30-0XB0)	SM 1231 AI 8 x 13 bit (6ES7 231-4HF30-0XB0)
	

Note

Unused analog inputs should be shorted.

When the inputs are configured for "current" mode, no current will flow through the input unless you supply external power to the module.

A.6.2 SM 1232 analog output module specifications

Table A- 79 General specifications

Technical data	SM 1232 AQ 2 x 14 bit	SM 1232 AQ 4 x 14 bit
Order number	6ES7 232-4HB30-0XB0	6ES7 232-4HD30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	45 x 100 x 75
Weight	180 grams	180 grams
Power dissipation	1.5 W	1.5 W
Current consumption (SM Bus)	80 mA	80 mA
Current consumption (24 VDC)	45 mA (no load)	45 mA (no load)

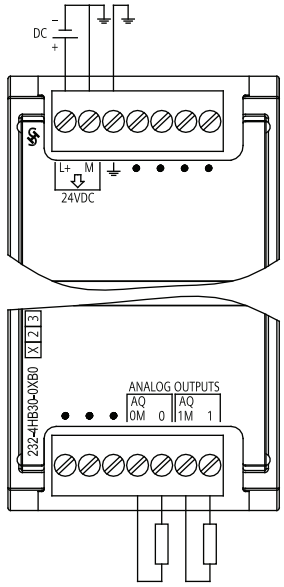
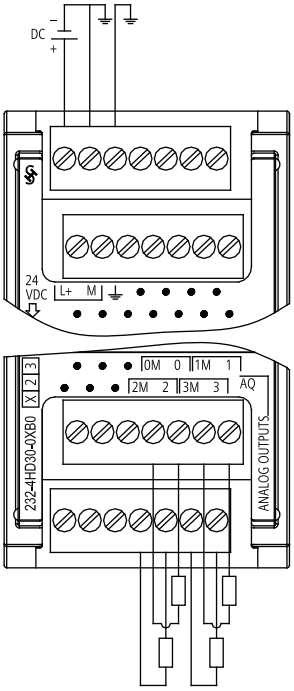
Table A- 80 Analog outputs

Technical data	SM 1232 AQ 2 x 14 bit	SM 1232 AQ 4 x 14 bit
Number of outputs	2	4
Type	Voltage or current	Voltage or current
Range	±10 V or 0 to 20 mA	±10 V or 0 to 20 mA
Resolution	Voltage: 14 bits Current: 13 bits	Voltage: 14 bits Current: 13 bits
Full scale range (data word)	Voltage: -27,648 to 27,648 ; Current: 0 to 27,648 Refer to the output ranges for voltage and current (Page 708).	
Accuracy (25°C / 0 to 55°C)	±0.3% / ±0.6% of full scale	
Settling time (95% of new value)	Voltage: 300 µS (R), 750 µS (1 uF) Current: 600 µS (1 mH), 2 ms (10 mH)	
Load impedance	Voltage: ≥ 1000 Ω Current: ≤ 600 Ω	
Behavior on RUN to STOP	Last value or substitute value (default value 0)	
Isolation (field side to logic)	none	
Cable length (meters)	100 m twisted and shielded	

Table A- 81 Diagnostics

Technical data	SM 1232 AQ 2 x 14 bit	SM 1232 AQ 4 x 14 bit
Overflow/underflow	Yes	Yes
Short to ground (voltage mode only)	Yes	Yes
Wire break (current mode only)	Yes	Yes
24 VDC low voltage	Yes	Yes

Table A- 82 Wiring diagrams for the analog output SMs

SM 1232 AQ 2 x 14 bit (6ES7 232-4HB30-0XB0)	SM 1232 AQ 4 x 14 bit (6ES7 232-4HD30-0XB0)
	

A.6.3 SM 1234 analog input/output module specifications

Table A- 83 General specifications

Technical data	SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit
Order number	6ES7 234-4HE30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75
Weight	220 grams
Power dissipation	2.0 W
Current consumption (SM Bus)	80 mA
Current consumption (24 VDC)	60 mA (no load)

Table A- 84 Analog inputs

Model	SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit
Number of inputs	4
Type	Voltage or Current (differential): Selectable in groups of 2
Range	±10 V, ±5 V, ±2.5 V, or 0 to 20 mA

Model	SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit
Full scale range (data word)	-27,648 to 27,648
Overshoot/undershoot range (data word)	Voltage: 32,511 to 27,649 / -27,649 to -32,512 Current: 32,511 to 27,649 / 0 to -4864 Refer to the section on input ranges for voltage and current (Page 708).
Overflow/underflow (data word)	Voltage: 32,767 to 32,512 / -32,513 to -32,768 Current: 32,767 to 32,512 / -4865 to -32,768 Refer to the section on input ranges for voltage and current (Page 708).
Resolution	12 bits + sign bit
Maximum withstand voltage/current	±35 V / ±40 mA
Smoothing	None, weak, medium, or strong Refer to the section on step response times (Page 707).
Noise rejection	400, 60, 50, or 10 Hz Refer to the section on sample rates (Page 707).
Input impedance	≥ 9 MΩ (voltage) / 280 Ω (current)
Isolation (field side to logic)	None
Accuracy (25°C / 0 to 55°C)	±0.1% / ±0.2% of full scale
Analog to digital conversion time	625 µs (400 Hz rejection)
Common mode rejection	40 dB, DC to 60 Hz
Operational signal range	Signal plus common mode voltage must be less than +12 V and greater than -12 V
Cable length (meters)	100 m, twisted and shielded

Table A- 85 Analog outputs

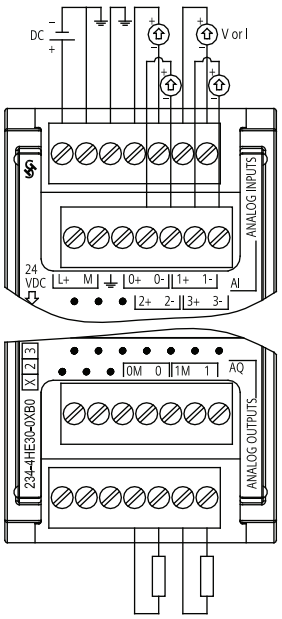
Technical data	SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit
Number of outputs	2
Type	Voltage or current
Range	±10 V or 0 to 20 mA
Resolution	Voltage: 14 bits ; Current: 13 bits
Full scale range (data word)	Voltage: -27,648 to 27,648 ; Current: 0 to 27,648 Refer to to the section on output ranges for voltage and current (Page 708).
Accuracy (25°C / 0 to 55°C)	±0.3% / ±0.6% of full scale
Settling time (95% of new value)	Voltage: 300 µS (R), 750 µS (1 uF) Current: 600 µS (1 mH), 2 ms (10 mH)
Load impedance	Voltage: ≥ 1000 Ω Current: ≤ 600 Ω
Behavior on RUN to STOP	Last value or substitute value (default value 0)
Isolation (field side to logic)	none
Cable length (meters)	100 m twisted and shielded

Table A- 86 Diagnostics

Model	SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit
Overflow/underflow	Yes [1]
Short to ground (voltage mode only)	Yes on outputs
Wire break (current mode only)	Yes on outputs
24 VDC low voltage	Yes

[1] If a voltage greater than +30 VDC or less than -15 VDC is applied to the input, the resulting value will be unknown and the corresponding overflow or underflow may not be active.

Table A- 87 Wiring diagrams for the analog input/output SM

SM 1234 AI 4 x 13 Bit / AQ 2 x 14 bit (6ES7 234-4HE30-0XB0)


Note

Unused analog inputs should be shorted.

When the inputs are configured for "current" mode, no current will flow through the input unless you supply external power to the module.

A.6.4 Step response of the analog inputs

Table A- 88 Step response (ms), 0 to full-scale measured at 95%

Smoothing selection (sample averaging)	Noise reduction/rejection frequency (Integration time selection)			
	400 Hz (2.5 ms)	60 Hz (16.6 ms)	50 Hz (20 ms)	10 Hz (100 ms)
None (1 cycle): No averaging	4 ms	18 ms	22 ms	100 ms
Weak (4 cycles): 4 samples	9 ms	52 ms	63 ms	320 ms
Medium (16 cycles): 16 samples	32 ms	203 ms	241 ms	1200 ms
Strong (32 cycles): 32 samples	61 ms	400 ms	483 ms	2410 ms
Sample time				
• 4 channels • 8 channels	• 0.625 ms • 1.25 ms	• 4.17 ms • 4.17 ms	• 5 ms • 5 ms	• 25 ms • 25 ms

A.6.5 Sample time and update times for the analog inputs

Table A- 89 Sample time and update time

Rejection frequency (Integration time)	Sample time	Update time for all channels	
		4-channel SM	8-channel SM
400 Hz (2.5 ms)	• 4-channel SM: 0.625 ms • 8-channel SM: 1.250 ms	2.5 ms	10 ms
60 Hz (16.6 ms)	4.170 ms	4.17 ms	4.17 ms
50 Hz (20 ms)	5.000 ms	5 ms	5 ms
10 Hz (100 ms)	25.000 ms	25 ms	25 ms

A.6.6 Measurement ranges of the analog inputs for voltage

Table A- 90 Analog input representation for voltage

System		Voltage Measuring Range					
Decimal	Hexadecimal	±10 V	±5 V	±2.5 V		0 to 10 V	
32767	7FFF	11.851 V	5.926 V	2.963 V	Overflow	11.851 V	Overflow
32512	7F00						
32511	7EFF	11.759 V	5.879 V	2.940 V	Overshoot range	11.759 V	Overshoot range
27649	6C01						
27648	6C00	10 V	5 V	2.5 V	Rated range	10 V	Rated range
20736	5100	7.5 V	3.75 V	1.875 V		7.5 V	
1	1	361.7 µV	180.8 µV	90.4 µV		361.7 µV	
0	0	0 V	0 V	0 V		0 V	
-1	FFFF					Negative values are not supported	
-20736	AF00	-7.5 V	-3.75 V	-1.875 V			
-27648	9400	-10 V	-5 V	-2.5 V			
-27649	93FF				Undershoot range		
-32512	8100	-11.759 V	-5.879 V	-2.940 V			
-32513	80FF				Underflow		
-32768	8000	-11.851 V	-5.926 V	-2.963 V			

A.6.7 Output (AQ) measurement ranges for voltage and current (SB and SM)

Table A- 91 Analog output representation for current

System		Current Output Range	
Decimal	Hexadecimal	0 mA to 20 mA	
32767	7FFF	See note 1	Overflow
32512	7F00	See note 1	
32511	7EFF	23.52 mA	Overshoot range
27649	6C01		
27648	6C00	20 mA	Rated range
20736	5100	15 mA	
1	1	723.4 nA	
0	0	0 mA	

[1] In an overflow or underflow condition, analog outputs will behave according to the device configuration properties set for the analog signal module. In the "Reaction to CPU STOP" parameter, select either: Use substitute value or Keep last value.

Table A- 92 Analog output representation for voltage

System		Voltage Output Range	
Decimal	Hexadecimal	± 10 V	
32767	7FFF	See note 1	Overflow
32512	7F00	See note 1	
32511	7EFF	11.76 V	Overshoot range
27649	6C01		
27648	6C00	10 V	Rated range
20736	5100	7.5 V	
1	1	361.7 µ V	
0	0	0 V	
-1	FFFF	-361.7 µ V	
-20736	AF00	-7.5 V	
-27648	9400	-10 V	
-27649	93FF		Undershoot range
-32512	8100	-11.76 V	
-32513	80FF	See note 1	Underflow
-32768	8000	See note 1	

1 In an overflow or underflow condition, analog outputs will behave according to the device configuration properties set for the analog signal module. In the "Reaction to CPU STOP" parameter, select either: Use substitute value or Keep last value.

A.7 Thermocouple and RTD signal modules (SMs)

A.7.1 SM 1231 Thermocouple

Table A- 93 General specifications

Model	SM 1231 AI 4 x 16 bit TC	SM 1231 AI 8 x 16 bit TC
Order number	6ES7 231-5QD30-0XB0	6ES7 231-5QF30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	45 x 100 x 75
Weight	180 grams	190 grams
Power dissipation	1.5 W	1.5 W
Current consumption (SM Bus)	80 mA	80 mA
Current consumption (24 VDC) [1]	40 mA	40 mA

1 20.4 to 28.8 VDC (Class 2, Limited Power, or sensor power from PLC)

Table A- 94 Analog inputs

Model		SM 1231 AI 4 x 16 bit TC	SM 1231 AI 8 x 16 bit TC
Number of inputs		4	8
Range Nominal range (data word) Overrange/underrange (data word) Overflow/underflow (data word)		See Thermocouple selection table (Page 712).	See Thermocouple selection table (Page 712).
Resolution	Temperature	0.1° C/0.1° F	0.1° C/0.1° F
	Voltage	15 bits plus sign	15 bits plus sign
Maximum withstand voltage		± 35 V	± 35 V
Noise rejection		85 dB for selected filter setting (10 Hz, 50 Hz, 60 Hz or 400 Hz)	85 dB for selected filter setting (10 Hz, 50 Hz, 60 Hz or 400 Hz)
Common mode rejection		> 120 dB at 120 VAC	> 120 dB at 120 VAC
Impedance		≥ 10 MΩ	≥ 10 MΩ
Isolation	Field to logic	500 VAC	500 VAC
	Field to 24 VDC	500 VAC	500 VAC
	24 VDC to logic	500 VAC	500 VAC
Channel to channel		120 VAC	120 VAC
Accuracy (25°C / 0 to 55°C)		See Thermocouple selection table (Page 712).	See Thermocouple selection table (Page 712).
Repeatability		±0.05% FS	±0.05% FS
Measuring principle		Integrating	Integrating
Module update time		See Noise reduction selection table (Page 712).	See Noise reduction selection table (Page 712).
Cold junction error		±1.5°C	±1.5°C
Cable length (meters)		100 meters to sensor max.	100 meters to sensor max.
Wire resistance		100 Ω max.	100 Ω max.

Table A- 95 Diagnostics

Model	SM 1231 AI 4 x 16 bit TC	SM 1231 AI 8 x 16 bit TC
Overflow/underflow [1]	Yes	Yes
Wire break (current mode only) [2]	Yes	Yes
24 VDC low voltage [1]	Yes	Yes

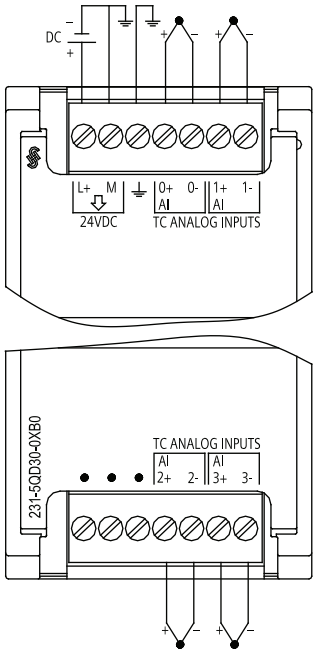
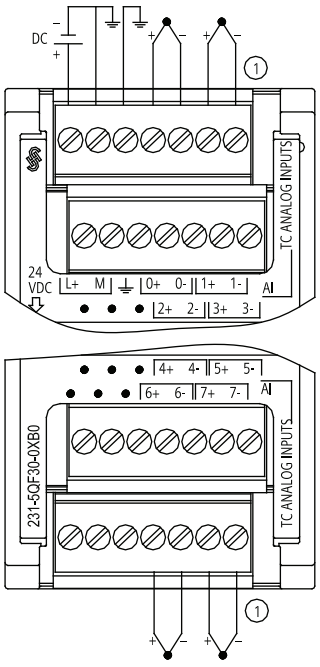
[1] The overflow, underflow and low voltage diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

[2] When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The SM 1231 Thermocouple (TC) analog signal module measures the value of voltage connected to the module inputs. The temperature measurement type can be either "Thermocouple" or "Voltage".

- "Thermocouple": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).
- "Voltage": The nominal range full scale value will be decimal 27648.

Table A- 96 Wiring diagrams for the thermocouple SMs

SM 1231 AI 4 x TC 16 bit (6ES7 231-5QD30-0XB0	SM 1231 AI 8 x TC bit (6ES7 231-5QF30-0XB0)
	

① TC 2, 3, 4, and 5 not shown connected for clarity.

Note

Unused analog inputs should be shorted.

The thermocouple unused channels can be deactivated. No error will occur if an unused channel is deactivated.

A.7.1.1 Basic operation for a thermocouple

Thermocouples are formed whenever two dissimilar metals are electrically bonded to each other. A voltage is generated that is proportional to the junction temperature. This voltage is small; one microvolt could represent many degrees. Measuring the voltage from a thermocouple, compensating for extra junctions, and then linearizing the result forms the basis of temperature measurement using thermocouples.

When you connect a thermocouple to the SM 1231 Thermocouple module, the two dissimilar metal wires are attached to the module at the module signal connector. The place where the two dissimilar wires are attached to each other forms the sensor thermocouple.

Two more thermocouples are formed where the two dissimilar wires are attached to the signal connector. The connector temperature causes a voltage that adds to the voltage from the sensor thermocouple. If this voltage is not corrected, then the temperature reported will deviate from the sensor temperature.

Cold junction compensation is used to compensate for the connector thermocouple. Thermocouple tables are based on a reference junction temperature, usually zero degrees Celsius. The cold junction compensation compensates the connector to zero degrees Celsius. The cold junction compensation restores the voltage added by the connector thermocouples. The temperature of the module is measured internally, then converted to a value to be added to the sensor conversion. The corrected sensor conversion is then linearized using the thermocouple tables.

For optimum operation of the cold junction compensation, the thermocouple module must be located in a thermally stable environment. Slow variation (less than 0.1° C/minute) in ambient module temperature is correctly compensated within the module specifications. Air movement across the module will also cause cold junction compensation errors.

If better cold junction error compensation is needed, an external iso-thermal terminal block may be used. The thermocouple module provides for use of a 0° C referenced or 50° C referenced terminal block.

A.7.1.2 Selection tables for the SM 1231 thermocouple

The ranges and accuracy for the different thermocouple types supported by the SM 1231 Thermocouple signal module are shown in the table below.

Table A- 97 SM 1231 Thermocouple selection table

Type	Under-range minimum[1]	Nominal range low limit	Nominal range high limit	Over-range maximum[2]	Normal range [3, 4] accuracy @ 25°C	Normal range [1, 2] accuracy 0°C to 55°C
J	-210.0°C	-150.0°C	1200.0°C	1450.0°C	±0.3°C	±0.6°C
K	-270.0°C	-200.0°C	1372.0°C	1622.0°C	±0.4°C	±1.0°C
T	-270.0°C	-200.0°C	400.0°C	540.0°C	±0.5°C	±1.0°C
E	-270.0°C	-200.0°C	1000.0°C	1200.0°C	±0.3°C	±0.6°C
R & S	-50.0°C	100.0°C	1768.0°C	2019.0°C	±1.0°C	±2.5°C
N	-270.0°C	-200.0°C	1300.0°C	1550.0°C	±1.0°C	±1.6°C
C	0.0°C	100.0°C	2315.0°C	2500.0°C	±0.7°C	±2.7°C

Type	Under-range minimum[1]	Nominal range low limit	Nominal range high limit	Over-range maximum[2]	Normal range [3, 4] accuracy @ 25°C	Normal range [1, 2] accuracy 0°C to 55°C
TXK/XK(L)	-200.0°C	-150.0°C	800.0°C	1050.0°C	±0.6°C	±1.2°C
Voltage	-32512	-27648 -80mV	27648 80mV	32511	±0.05%	±0.1%

1 Thermocouple values below the under-range minimum value are reported as -32768.

2 Thermocouple values above the over-range minimum value are reported as 32767.

3 Internal cold junction error is ±1.5°C for all ranges. This adds to the error in this table. The module requires at least 30 minutes of warmup time to meet this specification.

4 In the presence of radiated radio frequency of 970 MHz to 990 MHz, the accuracy of the SM 1231 AI 4 x 16 bit TC may be degraded.

Table A- 98 Noise reduction and update times for the SM 1231 Thermocouple

Rejection frequency selection	Integration time	4 Channel module update time (seconds)	8 Channel module update time (seconds)
400 Hz (2.5 ms)	10 ms [1]	0.143	0.285
60 Hz (16.6 ms)	16.67 ms	0.223	0.445
50 Hz (20 ms)	20 ms	0.263	0.525
10 Hz (100 ms)	100 ms	1.225	2.450

1 To maintain module resolution and accuracy when 400 Hz rejection is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

It is recommended for measuring thermocouples that a 100 ms integration time be used. The use of smaller integration times will increase the repeatability error of the temperature readings.

Note

After power is applied, the module performs internal calibration for the analog-to-digital converter. During this time the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time. Because the configuration of the module can vary the length of the initialization time, you should verify the behavior of the module in your configuration. If required, you can include logic in your user program to accommodate the initialization time of the module.

Representation of analog values for Thermocouple Type J

A representation of the analog values of thermocouples type J is shown in the table below.

Table A- 99 Representation of analog values of thermocouples type J

Type J in °C	Units		Type J in °F	Units		Range
	Decimal	Hexadecimal		Decimal	Hexadecimal	
> 1450.0	32767	7FFF	> 2642.0	32767	7FFF	Overflow
1450.0 : 1200.1	14500 : 12001	38A4 : 2EE1	2642.0 : 2192.2	26420 : 21922	6734 : 55A2	Overrange
1200.0 : -150.0	12000 : -1500	2EE0 : FA24	2192.0 : -238.0	21920 : -2380	55A0 : F6B4	Rated range
< -150.0	-32768	8000	< -238.0	-32768	8000	Underflow[1]

[1] Faulty wiring (for example, polarity reversal, or open inputs) or sensor error in the negative range (for example, wrong type of thermocouple) may cause the thermocouple module to signal underflow.

A.7.2 SM 1231 RTD

SM 1231 RTD specifications

Table A- 100 General specifications

Technical data	SM 1231 AI 4 x RTD x 16 bit	SM 1231 AI 8 x RTD x 16 bit
Order number	6ES7 231-5PD30-0XB0	6ES7 231-5PF30-0XB0
Dimensions W x H x D (mm)	45 x 100 x 75	70 x 100 x 75
Weight	220 grams	270 grams
Power dissipation	1.5 W	1.5 W
Current consumption (SM Bus)	80 mA	90 mA
Current consumption (24 VDC) [1]	40 mA	40 mA

[1] 20.4 to 28.8 VDC (Class 2, Limited Power, or sensor power from CPU)

Table A- 101 Analog inputs

Technical data	SM 1231 AI 4 x RTD x 16 bit	SM 1231 AI 8 x RTD x16 bit
Number of inputs	4	8
Type	Module referenced RTD and Ω	Module referenced RTD and Ω

Technical data		SM 1231 AI 4 x RTD x 16 bit	SM 1231 AI 8 x RTD x16 bit
Range Nominal range (data word) Overshoot/undershoot range (data word) Overflow/underflow (data word)		See RTD Sensor selection table (Page 717).	See RTD Sensor selection table (Page 717).
Resolution	Temperature	0.1° C/0.1° F	0.1° C/0.1° F
	Resistance	15 bits plus sign	15 bits plus sign
Maximum withstand voltage		± 35 V	± 35 V
Noise rejection		85 dB for the selected noise reduction (10 Hz, 50 Hz, 60 Hz or 400 Hz)	85 dB for the selected noise reduction (10 Hz, 50 Hz, 60 Hz or 400 Hz)
Common mode rejection		> 120dB	> 120dB
Impedance		≥ 10 MΩ	≥ 10 MΩ
Isolation	Field side to logic	500 VAC	500 VAC
	Field to 24 VDC	500 VAC	500 VAC
	24 VDC to logic	500 VAC	500 VAC
Channel to channel isolation		none	none
Accuracy		See RTD Sensor selection table (Page 717).	See RTD Sensor selection table (Page 717).
Repeatability		±0.05% FS	±0.05% FS
Maximum sensor dissipation		0.5m W	0.5m W
Measuring principle		Integrating	Integrating
Module update time		See Noise reduction selection table (Page 717).	See Noise reduction selection table (Page 717).
Cable length (meters)		100 meters to sensor max.	100 meters to sensor max.
Wire resistance		20 Ω, 2.7 Ω for 10 Ω RTD max.	20 Ω, 2.7 Ω for 10 Ω RTD max.

Table A- 102 Diagnostics

Technical data	SM 1231 AI 4 x RTD x 16 bit	SM 1231 AI 8 x RTD x16 bit
Overflow/underflow [1,2]	Yes	Yes
Wire break [3]	Yes	Yes
24 VDC low voltage [1]	Yes	Yes

1 The overflow, underflow and low voltage diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

2 For resistance ranges underflow detection is never enabled.

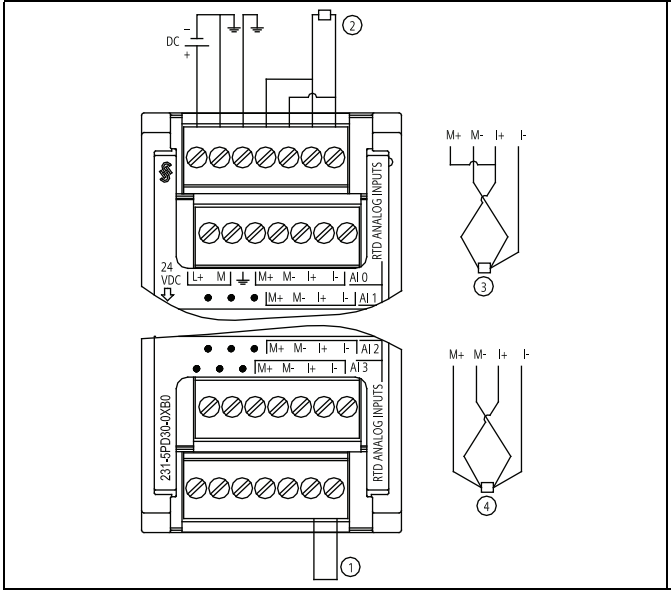
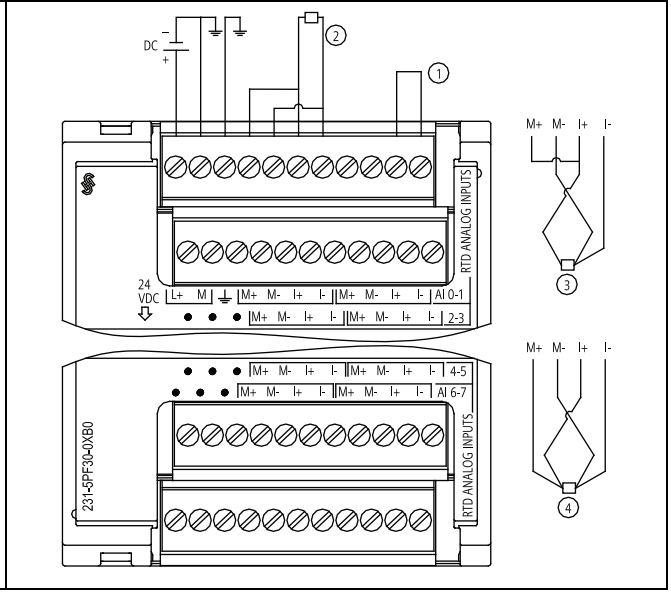
3 When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The SM 1231 RTD analog signal module measures the value of resistance connected to the module inputs. The measurement type can be selected as either "Resistor" or "Thermal resistor".

- "Resistor": The nominal range full scale value will be decimal 27648.
- "Thermal resistor": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).

The SM 1231 RTD module supports measurements with 2-wire, 3-wire and 4-wire connections to the sensor resistor.

Table A- 103 Wiring diagrams for the RTD SMs

SM 1231 RTD 4 x 16 bit (6ES7 231-5PD30-0XB)	SM 1231 RTD 8 x 16 bit (6ES7 231-5PF30-0XB0)
	

① Loop-back unused RTD inputs

② 2-wire RTD ③ 3-wire RTD ④ 4-wire RTD

Note

The RTD unused channels can be deactivated. No error will occur if an unused channel is deactivated.

The RTD module needs to have the current loop continuous to eliminate extra stabilization time which is automatically added to an unused channel that is not deactivated. For consistency the RTD module should have a resistor connected (like the 2-wire RTD connection).

A.7.2.1 Selection tables for the SM 1231 RTD

Table A- 104 Ranges and accuracy for the different sensors supported by the RTD modules

Temperature coefficient	RTD type	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range accuracy @ 25°C	Normal range accuracy 0°C to 55°C
Pt 0.003850	Pt 10	-243.0°C	-200.0°C	850.0°C	1000.0°C	±1.0°C	±2.0°C
ITS90 DIN EN 60751	Pt 50	-243.0°C	-200.0°C	850.0°C	1000.0°C	±0.5°C	±1.0°C
	Pt 100						
	Pt 200						
	Pt 500						
	Pt 1000						
Pt 0.003902	Pt 100	-243.0°C	-200.0°C	850.0°C	1000.0°C	± 0.5°C	±1.0°C
Pt 0.003916 Pt 0.003920	Pt 200	-243.0°C	-200.0°C	850.0°C	1000.0°C	± 0.5°C	±1.0°C
	Pt 500						
	Pt 1000						
Pt 0.003910	Pt 10	-273.2°C	-240.0°C	1100.0°C	1295°C	±1.0°C	±2.0°C
	Pt 50	-273.2°C	-240.0°C	1100.0°C	1295°C	±0.8°C	±1.6°C
	Pt 100						
	Pt 500						
Ni 0.006720 Ni 0.006180	Ni 100	-105.0°C	-60.0°C	250.0°C	295.0°C	±0.5°C	±1.0°C
	Ni 120						
	Ni 200						
	Ni 500						
	Ni 1000						
LG-Ni 0.005000	LG-Ni 1000	-105.0°C	-60.0°C	250.0°C	295.0°C	±0.5°C	±1.0°C
Ni 0.006170	Ni 100	-105.0°C	-60.0°C	180.0°C	212.4°C	±0.5°C	±1.0°C
Cu 0.004270	Cu 10	-240.0°C	-200.0°C	260.0°C	312.0°C	±1.0°C	±2.0°C
Cu 0.004260	Cu 10	-60.0°C	-50.0°C	200.0°C	240.0°C	±1.0°C	±2.0°C
	Cu 50	-60.0°C	-50.0°C	200.0°C	240.0°C	±0.6°C	±1.2°C
	Cu 100						
Cu 0.004280	Cu 10	-240.0°C	-200.0°C	200.0°C	240.0°C	±1.0°C	±2.0°C
	Cu 50	-240.0°C	-200.0°C	200.0°C	240.0°C	±0.7°C	±1.4°C
	Cu 100						

1 RTD values below the under-range minimum value report -32768.

2 RTD values above the over-range maximum value report -32767.

Table A- 105 Resistance

Range	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range accuracy @ 25°C	Normal range accuracy 0°C to 55°C
150 Ω	n/a	0 (0 Ω)	27648 (150 Ω)	176.383 Ω	±0.05%	±0.1%
300 Ω	n/a	0 (0 Ω)	27648 (300 Ω)	352.767 Ω	±0.05%	±0.1%
600 Ω	n/a	0 (0 Ω)	27648 (600 Ω)	705.534 Ω	±0.05%	±0.1%

1 RTD values below the under-range minimum value are reported as -32768.

2 RTD values above the over-range minimum value are reported as -32768.

Note

The module reports 32767 on any activated channel with no sensor connected. If open wire detection is also enabled, the module flashes the appropriate red LEDs.

When 500 Ω and 1000 Ω RTD ranges are used with other lower value resistors, the error may increase to two times the specified error.

Best accuracy will be achieved for the 10 Ω RTD ranges if 4 wire connections are used.

The resistance of the connection wires in 2 wire mode will cause an error in the sensor reading and therefore accuracy is not guaranteed.

Table A- 106 Noise reduction and update times for the RTD modules

Rejection frequency selection	Integration time	Update time (seconds)	
		4-channel module	8-channel module
400 Hz (2.5 ms)	10 ms [1]	4-/2-wire: 0.142 3-wire: 0.285	4-/2-wire: 0.285 3-wire: 0.525
60 Hz (16.6 ms)	16.67 ms	4-/2-wire: 0.222 3-wire: 0.445	4-/2-wire: 0.445 3-wire: 0.845
50 Hz (20 ms)	20 ms	4-/2-wire: 0.262 3-wire: .505	4-/2-wire: 0.524 3-wire: 1.015
10 Hz (100 ms)	100 ms	4-/2-wire: 1.222 3-wire: 2.445	4-/2-wire: 2.425 3-wire: 4.845

1 To maintain module resolution and accuracy when the 400 Hz filter is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

NOTICE
After power is applied, the module performs internal calibration for the analog-to-digital converter. During this time the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time. Because the configuration of the module can vary the length of the initialization time, you should verify the behavior or the module in your configuration. If required, you can include logic in your user program to accommodate the initialization time of the module.

Representation of Analog values for RTDs

A representation of the digitized measured value for the RTD standard temperature range sensors are shown in the tables below.

Table A- 107 Representation of analog values for resistance thermometers PT 100, 200, 500, 1000 and PT 10, 50, 100, 500 GOST (0.003850) standard

Pt x00 standard in °C (1 digit = 0.1° C)	Units		Pt x00 standard in °F (1 digit = 0.1 F)	Units		Range
	Decimal	Hexadecimal		Decimal	Hexadecimal	
> 1000.0	32767	7FFF	> 1832.0	32767	7FFF	Overflow
1000.0 : 850.1	10000 : 8501	2710 : 2135	1832.0 : 1562.1	18320 : 15621	4790 : 3D05	Overrange
850.0 : -200.0	8500 : -2000	2134 : F830	1562.0 : -328.0	15620 : -3280	3D04 : F330	Rated range
-200.1 : -243.0	-2001 : -2430	F82F : F682	-328.1 : -405.4	-3281 : -4054	F32F : F02A	Underrange
< -243.0	-32768	8000	< -405.4	-32768	8000	Underflow

A.8 Digital signal boards (SBs)

A.8.1 SB 1221 200 kHz digital input specifications

Table A- 108 General specifications

Technical data	SB 1221 DI 4 x 24 VDC, 200 kHz	SB 1221 DI 4 x 5 VDC, 200 kHz
Order number	6ES7 221-3BD30-0XB0	6ES7 221-3AD30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21	38 x 62 x 21
Weight	35 grams	35 grams
Power dissipation	1.5 W	1.0 W
Current consumption (SM Bus)	40 mA	40 mA
Current consumption (24 VDC)	7 mA / input + 20 mA	15 mA / input + 15 mA

Table A- 109 Digital inputs

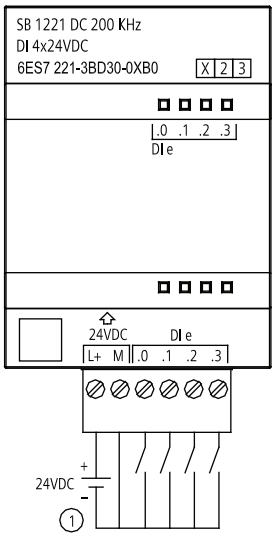
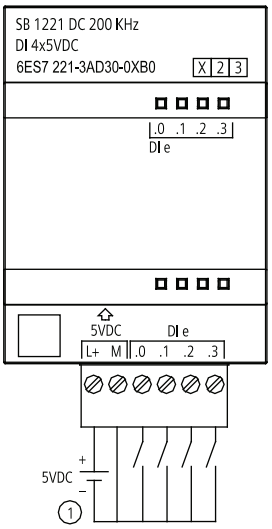
Technical data	SB 1221 DI 4 x 24 VDC, 200 kHz	SB 1221 DI 4 x 5 VDC, 200 kHz
Number of inputs	4	4
Type	Source	Source
Rated voltage	24 VDC at 7 mA, nominal	5 VDC at 15 mA, nominal
Continuous permissible voltage	28.8 VDC	6 VDC
Surge voltage	35 VDC for 0.5 sec.	6 V
Logic 1 signal (min.)	L+ minus 10 VDC at 2.9 mA	L+ minus 2.0 VDC at 5.1 mA
Logic 0 signal (max.)	L+ minus 5 VDC at 1.4 mA	L+ minus 1.0 VDC at 2.2 mA
HSC clock input rates (max.)	Single phase: 200 kHz Quadrature phase: 160 kHz	Single phase: 200 kHz Quadrature phase: 160 kHz
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	1	1
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms; Selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms; Selectable in groups of 4
Number of inputs on simultaneously	4	4
Cable length (meters)	50 shielded twisted pair	50 shielded twisted pair

NOTICE

When switching frequencies above 20 kHz, it is important that the digital inputs receive a square wave. Consider the following options to improve the signal quality to the inputs:

- Minimize the cable length
- Change a driver from a sink only driver to a sinking and sourcing driver
- Change to a higher quality cable
- Reduce the circuit/components from 24 V to 5 V
- Add an external load at the input

Table A- 110 Wiring diagrams for the 200 kHz digital input SBs

SB 1221 DI 4 x 24 VDC, 200 kHz (6ES7 221-3BD30-0XB0)	SB 1221 DI 4 x 5 VDC, 200 kHz (6ES7 221-3AD30-0XB0)
	

① Supports sourcing inputs only

A.8.2 SB 1222 200 kHz digital output specifications

Table A- 111 General specifications

Technical data	SB 1222 DQ 4 x 24 VDC, 200 kHz	SB 1222 DQ 4 x 5 VDC, 200 kHz
Order number	6ES7 222-1BD30-0XB0	6ES7 222-1AD30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21	38 x 62 x 21
Weight	35 grams	35 grams
Power dissipation	0.5 W	0.5 W
Current consumption (SM Bus)	35 mA	35 mA
Current consumption (24 VDC)	15 mA	15 mA

Table A- 112 Digital outputs

Technical data	SB 1222 DQ 4 x 24 VDC, 200 kHz	SB 1222 DQ 4 x 5 VDC, 200 kHz
Number of outputs	4	4
Output type	Solid state - MOSFET sink and source[1]	Solid state - MOSFET sink and source[1]
Voltage range	20.4 to 28.8 VDC	4.25 to 6.0 VDC
Logic 1 signal at max. current	L+ minus 1.5 V	L+ minus 0.7 V
Logic 0 signal at max. current	1.0 VDC, max.	0.2 VDC, max.
Current (max.)	0.1 A	0.1 A
Lamp load	--	--
On state contact resistance	11 Ω max.	7 Ω max.
Off state resistance	6 Ω max.	0.2 Ω max.
Leakage current per point	--	--
Pulse Train Output rate	200 kHz max., 2 Hz min.	200 kHz max., 2 Hz min.
Surge current	0.11 A	0.11 A
Overload protection	No	No
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	1	1
Currents per common	0.4 A	0.4 A
Inductive clamp voltage	None	None
Switching delay	1.5 µs + 300 ns rise 1.5 µs + 300 ns fall	200 ns + 300 ns rise 200 ns + 300 ns fall
Behavior on RUN to STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	4	4
Cable length (meters)	50 shielded twisted pair	50 shielded twisted pair

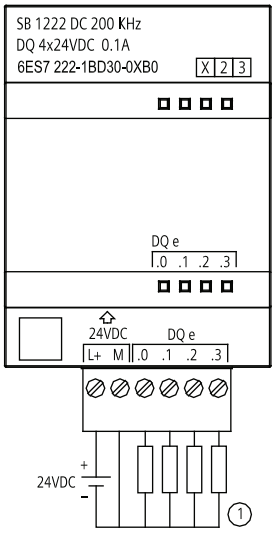
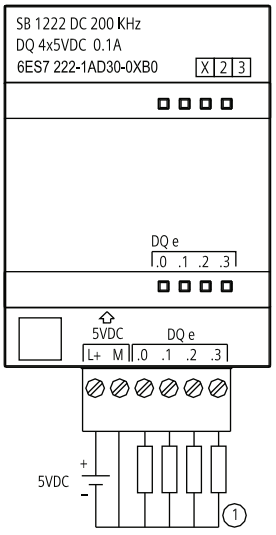
[1] Because both sinking and sourcing configurations are supported by the same circuitry, the active state of a sourcing load is opposite that of a sinking load. A source output exhibits positive logic (Q bit and LED are ON when the load has current flow), while a sink output exhibits negative logic (Q bit and LED are OFF when the load has current flow). If the module is plugged in with no user program, the default for this module is 0 V, which means that a sinking load will be turned ON.

NOTICE

When switching frequencies above 20 kHz, it is important that the digital inputs receive a square wave. Consider the following options to improve the signal quality to the inputs:

- Minimize the cable length
- Change a driver from a sink only driver to a sinking and sourcing driver
- Change to a higher quality cable
- Reduce the circuit/components from 24 V to 5 V
- Add an external load at the input

Table A- 113 Wiring diagrams for the 200 kHz digital output SBs

SB 1222 DQ 4 x 24 VDC, 200 kHz (6ES7 222-1BD30-0XB0)	SB 1222 DQ 4 x 5 VDC, 200 kHz (6ES7 222-1AD30-0XB0)
	

① For sourcing outputs, connect "Load" to "-" (shown). For sinking outputs, connect "Load" to "+". Because both sinking and sourcing configurations are supported by the same circuitry, the active state of a sourcing load is opposite that of a sinking load. A source output exhibits positive logic (Q bit and LED are ON when the load has current flow), while a sink output exhibits negative logic (Q bit and LED are OFF when the load has current flow). If the module is plugged in with no user program, the default for this module is 0 V, which means that a sinking load will be turned ON.

A.8.3 SB 1223 200 kHz digital input / output specifications

Table A- 114 General specifications

Technical data	SB 1223 DI 2 x 24 VDC / DQ 2 x 24 VDC, 200 kHz	SB 1223 DI 2 x 5 VDC / DQ 2 x 5 VDC, 200 kHz
Order number	6ES7 223-3BD30-0XB0	6ES7 223-3AD30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21	38 x 62 x 21
Weight	35 grams	35 grams
Power dissipation	1.0 W	0.5 W
Current consumption (SM Bus)	35 mA	35 mA
Current consumption (24 VDC)	7 mA / Input + 30 mA	15 mA / input + 15 mA

Table A- 115 Digital inputs

Technical data	SB 1223 DI 2 x 24 VDC / DQ 2 x 24 VDC, 200 kHz	SB 1223 DI 2 x 5 VDC / DQ 2 x 5 VDC, 200 kHz
Number of inputs	2	2
Type	Source	Source
Rated voltage	24 VDC at 7 mA, nominal	5 VDC at 15 mA, nominal
Continuous permissible voltage	28.8 VDC	6 VDC
Surge voltage	35 VDC for 0.5 sec.	6 V
Logic 1 signal (min.)	L+ minus 10 VDC at 2.9 mA	L+ minus 2.0 VDC at 5.1 mA
Logic 0 signal (max.)	L+ minus 5 VDC at 1.4 mA	L+ minus 1.0 VDC at 2.2 mA
HSC clock input rates (max.)	Single phase: 200 kHz Quadrature phase: 160 kHz	Single phase: 200 kHz Quadrature phase: 160 kHz
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	1 (no isolation to outputs)	1 (no isolation to outputs)
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms; Selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms; Selectable in groups of 4
Number of inputs on simultaneously	2	2
Cable length (meters)	50 shielded twisted pair	50 shielded twisted pair

Table A- 116 Digital outputs

Technical data	SB 1223 DI 2 x 24 VDC / DQ 2 x 24 VDC, 200 kHz	SB 1223 DI 2 x 5 VDC / DQ 2 x 5 VDC, 200 kHz
Number of outputs	2	2
Output type	Solid state - MOSFET sink and source[1]	Solid state - MOSFET sink and source[1]
Voltage range	20.4 to 28.8 VDC	4.25 to 6.0 VDC
Rated value	24 VDC	5 VDC
Logic 1 signal at max. current	L+ minus 1.5 V	L+ minus 0.7 V
Logic 0 signal at max. current	1.0 VDC, max.	0.2 VDC, max.

Technical data	SB 1223 DI 2 x 24 VDC / DQ 2 x 24 VDC, 200 kHz	SB 1223 DI 2 x 5 VDC / DQ 2 x 5 VDC, 200 kHz
Current (max.)	0.1 A	0.1 A
Lamp load	--	--
On state contact resistance	11 Ω max.	7 Ω max.
Off state resistance	6 Ω max.	0.2 Ω max.
Leakage current per point	--	--
Pulse Train Output rate	200 kHz max., 2 Hz min.	200 kHz max., 2 Hz min.
Surge current	0.11 A	0.11 A
Overload protection	No	No
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	1 (no isolation to inputs)	1 (no isolation to inputs)
Currents per common	0.2 A	0.2 A
Inductive clamp voltage	None	None
Switching delay	1.5 µs + 300 ns rise 1.5 µs + 300 ns fall	200 ns + 300 ns rise 200 ns + 300 ns fall
Behavior on RUN to STOP	Last value or substitute (default value 0)	Last value or substitute (default value 0)
Number of outputs on simultaneously	2	2
Cable length (meters)	50 shielded twisted pair	50 shielded twisted pair

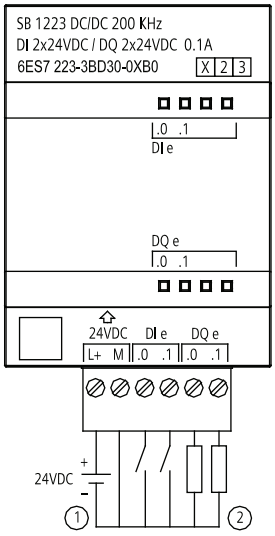
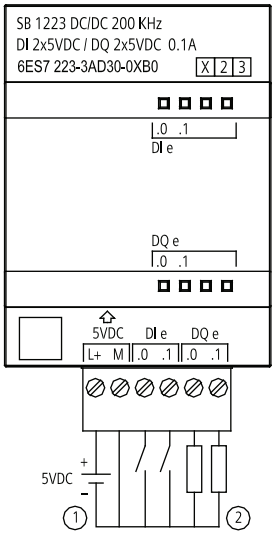
[1] Because both sinking and sourcing configurations are supported by the same circuitry, the active state of a sourcing load is opposite that of a sinking load. A source output exhibits positive logic (Q bit and LED are ON when the load has current flow), while a sink output exhibits negative logic (Q bit and LED are OFF when the load has current flow). If the module is plugged in with no user program, the default for this module is 0 V, which means that a sinking load will be turned ON.

NOTICE

When switching frequencies above 20 kHz, it is important that the digital inputs receive a square wave. Consider the following options to improve the signal quality to the inputs:

- Minimize the cable length
- Change a driver from a sink only driver to a sinking and sourcing driver
- Change to a higher quality cable
- Reduce the circuit/components from 24 V to 5 V
- Add an external load at the input

Table A- 117 Wiring diagrams for the 200 kHz digital input/output SBs

SB 1223 DI 2 x 24 VDC/DQ 2 x 24 VDC, 200 kHz (6ES7 223-3BD30-0XB0)	SB 1223 DI 2 x 5 VDC / DQ 2 x 5 VDC, 200 kHz (6ES7 223-3AD30-0XB0)
	

① Supports sourcing inputs only

② For sourcing outputs, connect "Load" to "-" (shown). For sinking outputs, connect "Load" to "+". [1] Because both sinking and sourcing configurations are supported by the same circuitry, the active state of a sourcing load is opposite that of a sinking load. A source output exhibits positive logic (Q bit and LED are ON when the load has current flow), while a sink output exhibits negative logic (Q bit and LED are OFF when the load has current flow). If the module is plugged in with no user program, the default for this module is 0 V, which means that a sinking load will be turned ON.

A.8.4 SB 1223 2 X 24 VDC input / 2 X 24 VDC output specifications

Table A- 118 General specifications

Technical Data	SB 1223 DI 2 x 24 VDC, DQ 2 x 24 VDC
Order number	6ES7 223-0BD30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21
Weight	40 grams
Power dissipation	1.0 W
Current consumption (SM Bus)	50 mA
Current consumption (24 VDC)	4 mA / Input used

Table A- 119 Digital inputs

Technical Data	SB 1223 DI 2 x 24 VDC, DQ 2 x 24 VDC
Number of inputs	2
Type	IEC Type 1 sink

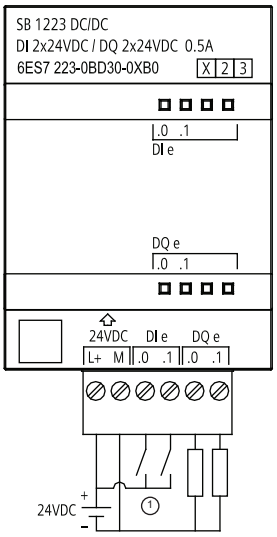
Technical Data	SB 1223 DI 2 x 24 VDC, DQ 2 x 24 VDC
Rated voltage	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.
Logic 1 signal (min.)	15 VDC at 2.5 mA
Logic 0 signal (max.)	5 VDC at 1 mA
HSC clock input rates (max.)	20 kHz (15 to 30 VDC) 30 kHz (15 to 26 VDC)
Isolation (field side to logic)	500 VAC for 1 minute
Isolation groups	1
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms Selectable in groups of 2
Number of inputs on simultaneously	2
Cable length (meters)	500 shielded, 300 unshielded

Table A- 120 Digital outputs

Technical Data	SB 1223 DI 2 x 24 VDC, DQ 2 x 24 VDC
Number of outputs	2
Output type	Solid state - MOSFET (sourcing)
Voltage range	20.4 to 28.8 VDC
Logic 1 signal at max. current	20 VDC min.
Logic 0 signal with 10K Ω load	0.1 VDC max.
Current (max.)	0.5 A
Lamp load	5 W
On state contact resistance	0.6 Ω max.
Leakage current per point	10 µA max.
Pulse Train Output (PTO) rate	20 KHz max., 2 Hz min. [1]
Surge current	5 A for 100 ms max.
Overload protection	No
Isolation (field side to logic)	500 VAC for 1 minute
Isolation groups	1
Currents per common	1 A
Inductive clamp voltage	L+ minus 48 V, 1 W dissipation
Switching delay	2 µs max. off to on 10 µs max. on to off
Behavior on RUN to STOP	Last value or substitute value (default value 0)
Number of outputs on simultaneously	2
Cable length (meters)	500 m shielded, 150 m unshielded

[1] Depending on your pulse receiver and cable, an additional load resistor (at least 10% of rated current) may improve pulse signal quality and noise immunity.

Table A- 121 Wiring diagram for the digital input/output SB

SB 1223 DI 2 x 24 VDC, DQ 2 x 24 VDC (6ES7 223-0BD30-0XB0)


① Supports sinking inputs only

A.9 Analog signal boards (SBs)

A.9.1 SB 1231 1 analog input specifications

Note

To use this SB, your CPU firmware must be V2.0 or higher.

Table A- 122 General specifications

Technical data	SB 1231 AI 1 x 12 bit
Order number	6ES7 231-4HA30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21 mm
Weight	35 grams
Power dissipation	0.4 W
Current consumption (SM Bus)	55 mA
Current consumption (24 VDC)	none

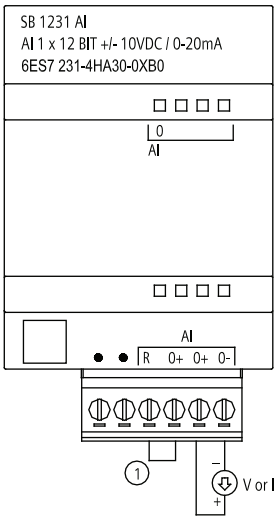
Table A- 123 Analog inputs

Technical data	SB 1231 AI 1x12 bit
Number of inputs	1
Type	Voltage or current (differential)
Range	±10V, ±5V, ±2.5 or 0 to 20 mA
Resolution	11 bits + sign bit
Full scale range (data word)	-27,648 to 27,648
Over/Under range (data word)	Voltage: 32,511 to 27,649 / -27,649 to -32,512 Current: 32,511 to 27,649 / 0 to -4,864 (Refer to Analog input representation for voltage and Analog input representation for current (Page 733).)
Overflow/Underflow (data word)	Voltage: 32,767 to 32,512 / -32,513 to -32,768 Current: 32,767 to 32,512 / -4,865 to -32,768 (Refer to Analog input representation for voltage and Analog input representation for current (Page 733).)
Maximum withstand voltage / current	±35V / ±40 mA
Smoothing	None, weak, medium, or strong (refer to Analog input response times for step response time (Page 733).)
Noise rejection	400, 60, 50, or 10 Hz (refer to Analog input response times for sample rates (Page 733).)
Accuracy (25°C / 0 to 55°C)	±0.3% / ±0.6% of full scale
Input impedance Differential Common mode	 Voltage: 220 kΩ; Current: 250 Ω Voltage: 55 kΩ; Current: 55 kΩ
Behavior on RUN to STOP	Last value or substitute value (default value 0)
Measuring principle	Actual value conversion
Common mode rejection	400 dB, DC to 60 Hz
Operational signal range	Signal plus common mode voltage must be less than +35 V and greater than -35 V
Isolation (field side to logic)	None
Cable length (meters)	100 m, twisted and shielded

Table A- 124 Diagnostics

Technical data	SB 1231 AI 1 x 12 bit
Overflow/underflow	Yes
24 VDC low voltage	no

Table A- 125 Wiring diagram for the analog input SB

SB 1231 AI x 12 bit (6ES7 231-4HA30-0XB0)

① Connect "R" and "0+" for current

A.9.2 SB 1232 1 analog output specifications

Table A- 126 General specifications

Technical data	SB 1232 AQ 1 x 12 bit
Order number	6ES7 232-4HA30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21 mm
Weight	40 grams
Power dissipation	1.5 W
Current consumption (SM Bus)	15 mA
Current consumption (24 VDC)	40 mA (no load)

Table A- 127 Analog outputs

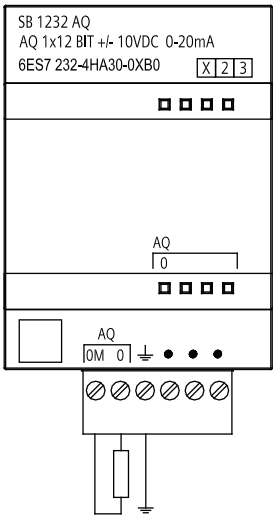
Technical data	SB 1232 AQ 1 x 12 bit
Number of outputs	1
Type	Voltage or current
Range	±10 V or 0 to 20 mA
Resolution	Voltage: 12 bits Current: 11 bits

Technical data	SB 1232 AQ 1 x 12 bit
Full scale range (data word) Refer to the output ranges for voltage and current (Page 733).	Voltage: -27,648 to 27,648 Current: 0 to 27,648
Accuracy (25°C / 0 to 55°C)	±0.5% / ±1% of full scale
Settling time (95% of new value)	Voltage: 300 µS (R), 750 µS (1 uF) Current: 600 µS (1 mH), 2 ms (10 mH)
Load impedance	Voltage: ≥ 1000 Ω Current: ≤ 600 Ω
Behavior on RUN to STOP	Last value or substitute value (default value 0)
Isolation (field side to logic)	None
Cable length (meters)	100 m, twisted and shielded

Table A- 128 Diagnostics

Technical data	SB 1232 AQ 1 x 12 bit
Overflow/underflow	Yes
Short to ground (voltage mode only)	Yes
Wire break (current mode only)	Yes

Table A- 129 Wiring diagram for the analog output SB

SB 1232 AQ 1 x 12 bit (6ES7 232-4HA30-0XB0)


A.9.3 Measurement ranges for analog inputs and outputs

A.9.3.1 Step response of the analog inputs

Table A- 130 Step response (ms), 0V to 10V measured at 95%

Smoothing selection (sample averaging)	Integration time selection			
	400 Hz (2.5 ms)	60 Hz (16.6 ms)	50 Hz (20 ms)	10 Hz (100 ms)
None (1 cycle): No averaging	4.5 ms	18.7 ms	22.0 ms	102 ms
Weak (4 cycles): 4 samples	10.6 ms	59.3 ms	70.8 ms	346 ms
Medium (16 cycles): 16 samples	33.0 ms	208 ms	250 ms	1240 ms
Strong (32 cycles): 32 samples	63.0 ms	408 ms	490 ms	2440 ms
Sample time	**0.156 ms**	**1.042 ms**	**1.250 ms**	**6.250 ms**

A.9.3.2 Sample time and update times for the analog inputs

Table A- 131 Sample time and update time

Selection	Sample time	SB update time
400 Hz (2.5 ms)	0.156 ms	0.156 ms
60 Hz (16.6 ms)	1.042 ms	1.042 ms
50 Hz (20 ms)	1.250 ms	1.25 ms
10 Hz (100 ms)	6.250 ms	6.25 ms

A.9.3.3 Measurement ranges of the analog inputs for voltage

Table A- 132 Analog input representation for voltage

System		Voltage Measuring Range					
Decimal	Hexadecimal	±10 V	±5 V	±2.5 V		0 to 10 V	
32767	7FFF	11.851 V	5.926 V	2.963 V	Overflow	11.851 V	Overflow
32512	7F00						
32511	7EFF	11.759 V	5.879 V	2.940 V	Overshoot range	11.759 V	Overshoot range
27649	6C01						
27648	6C00	10 V	5 V	2.5 V	Rated range	10 V	Rated range
20736	5100	7.5 V	3.75 V	1.875 V		7.5 V	
1	1	361.7 µV	180.8 µV	90.4 µV		361.7 µV	
0	0	0 V	0 V	0 V		0 V	
-1	FFFF					Negative values are not supported	
-20736	AF00	-7.5 V	-3.75 V	-1.875 V			
-27648	9400	-10 V	-5 V	-2.5 V			
-27649	93FF				Undershoot range		
-32512	8100	-11.759 V	-5.879 V	-2.940 V			
-32513	80FF				Underflow		
-32768	8000	-11.851 V	-5.926 V	-2.963 V			

A.9.3.4 Output (AQ) measurement ranges for voltage and current (SB and SM)

Table A- 133 Analog output representation for current

System		Current Output Range	
Decimal	Hexadecimal	0 mA to 20 mA	
32767	7FFF	See note 1	Overflow
32512	7F00	See note 1	
32511	7EFF	23.52 mA	Overshoot range
27649	6C01		
27648	6C00	20 mA	Rated range
20736	5100	15 mA	
1	1	723.4 nA	
0	0	0 mA	

1 In an overflow or underflow condition, analog outputs will behave according to the device configuration properties set for the analog signal module. In the "Reaction to CPU STOP" parameter, select either: Use substitute value or Keep last value.

Table A- 134 Analog output representation for voltage

System		Voltage Output Range	
Decimal	Hexadecimal	± 10 V	
32767	7FFF	See note 1	Overflow
32512	7F00	See note 1	
32511	7EFF	11.76 V	Overshoot range
27649	6C01		
27648	6C00	10 V	Rated range
20736	5100	7.5 V	
1	1	361.7 µ V	
0	0	0 V	
-1	FFFF	-361.7 µ V	
-20736	AF00	-7.5 V	
-27648	9400	-10 V	
-27649	93FF		Undershoot range
-32512	8100	-11.76 V	
-32513	80FF	See note 1	Underflow
-32768	8000	See note 1	

1 In an overflow or underflow condition, analog outputs will behave according to the device configuration properties set for the analog signal module. In the "Reaction to CPU STOP" parameter, select either: Use substitute value or Keep last value.

A.9.4 Thermocouple SBs

A.9.4.1 SB 1231 1 analog thermocouple input specifications

Note

To use this SB, your CPU firmware must be V2.0 or higher.

Table A- 135 General specifications

Technical data	SB 1231 AI 1 x 16 bit Thermocouple
Order number	6ES7 231-5QA30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21 mm
Weight	35 grams
Power dissipation	0.5 W
Current consumption (SM Bus)	5 mA
Current consumption (24 VDC)	20 mA

Table A- 136 Analog inputs

Technical data		SB 1231 AI 1x16 bit Thermocouple
Number of inputs		1
Type		Floating TC and mV
Range • Nominal range (data word) • Overrange/underrange (data word) • Overflow/underflow (data word)		See Thermocouple filter selection table (Page 736).
Resolution	Temperature	0.1° C / 0.1° F
	Voltage	15 bits plus sign
Maximum withstand voltage		±35 V
Noise rejection		85 dB for the selected filter setting (10 Hz, 50 Hz, 60 Hz, 400 Hz)
Common mode rejection		> 120 dB at 120 VAC
Impedance		≥ 10 M Ω
Accuracy (25°C / 0 to 55°C)		See Thermocouple selection table (Page 736).
Repeatability		±0.05% FS
Measuring principle		Integrating
Module update time		See Thermocouple filter selection table (Page 736).
Cold junction error		±1.5° C
Isolation (field side to logic)		500 VAC
Cable length (meters)		100 m to sensor max.
Wire resistance		100 Ω max.

Table A- 137 Diagnostics

Technical data	SB 1231 AI 1 x 16 bit Thermocouple
Overflow/underflow[1]	Yes
Wire break[2]	Yes

1 The overflow and underflow diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

2 When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The SM 1231 Thermocouple (TC) analog signal module measures the value of voltage connected to the module inputs.

The SB 1231 Thermocouple analog signal board measures the value of voltage connected to the signal board inputs. The temperature measurement type can be either "Thermocouple" or "Voltage".

- "Thermocouple": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).
- "Voltage": The nominal range full scale value will be decimal 27648.

A.9.4.2 Basic operation for a thermocouple

Thermocouples are formed whenever two dissimilar metals are electrically bonded to each other. A voltage is generated that is proportional to the junction temperature. This voltage is small; one microvolt could represent many degrees. Measuring the voltage from a thermocouple, compensating for extra junctions, and then linearizing the result forms the basis of temperature measurement using thermocouples.

When you connect a thermocouple to the SM 1231 Thermocouple module, the two dissimilar metal wires are attached to the module at the module signal connector. The place where the two dissimilar wires are attached to each other forms the sensor thermocouple.

Two more thermocouples are formed where the two dissimilar wires are attached to the signal connector. The connector temperature causes a voltage that adds to the voltage from the sensor thermocouple. If this voltage is not corrected, then the temperature reported will deviate from the sensor temperature.

Cold junction compensation is used to compensate for the connector thermocouple. Thermocouple tables are based on a reference junction temperature, usually zero degrees Celsius. The cold junction compensation compensates the connector to zero degrees Celsius. The cold junction compensation restores the voltage added by the connector thermocouples. The temperature of the module is measured internally, then converted to a value to be added to the sensor conversion. The corrected sensor conversion is then linearized using the thermocouple tables.

For optimum operation of the cold junction compensation, the thermocouple module must be located in a thermally stable environment. Slow variation (less than 0.1° C/minute) in ambient module temperature is correctly compensated within the module specifications. Air movement across the module will also cause cold junction compensation errors.

If better cold junction error compensation is needed, an external iso-thermal terminal block may be used. The thermocouple module provides for use of a 0° C referenced or 50° C referenced terminal block.

Selection table for the SB 1231 thermocouple

The ranges and accuracy for the different thermocouple types supported by the SB 1231 Thermocouple signal board are shown in the table below.

Table A- 138 SB 1231 Thermocouple selection table

Thermocouple Type	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range [3] accuracy @ 25°C	Normal range [3], accuracy 0°C to 55°C
J	-210.0°C	-150.0°C	1200.0°C	1450.0°C	±0.3°C	±0.6°C
K	-270.0°C	-200.0°C	1372.0°C	1622.0°C	±0.4°C	±1.0°C
T	-270.0°C	-200.0°C	400.0°C	540.0°C	±0.5°C	±1.0°C
E	-270.0°C	-200.0°C	1000.0°C	1200.0°C	±0.3°C	±0.6°C
R & S	-50.0°C	100.0°C	1768.0°C	2019.0°C	±1.0°C	±2.5°C
N	-270.0°C	0.0°C	1300.0°C	1550.0°C	±1.0°C	±1.6°C
C	0.0°C	100.0°C	2315.0°C	2500.0°C	±0.7°C	±2.7°C

Thermocouple Type	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range [3] accuracy @ 25°C	Normal range [3,] accuracy 0°C to 55°C
TXK/XK(L)	-200.0°C	-150.0°C	800.0°C	1050.0°C	±0.6°C	±1.2°C
Voltage	-32511	-27648 -80mV	27648 80mV	32511	±0.05%	±0.1%

1 Thermocouple values below the under-range minimum value are reported as -32768.

2 Thermocouple values above the over-range minimum value are reported as 32767.

3 Internal cold junction error is ±1.5°C for all ranges. This adds to the error in this table. The signal board requires at least 30 minutes of warmup time to meet this specification.

Table A- 139 Filter selection table for the SB 1231 Thermocouple

Rejection frequency (Hz)	Integration time (ms)	Signal board update time (seconds)
10	100	0.306
50	20	0.066
60	16.67	0.056
400[1]	10	0.036

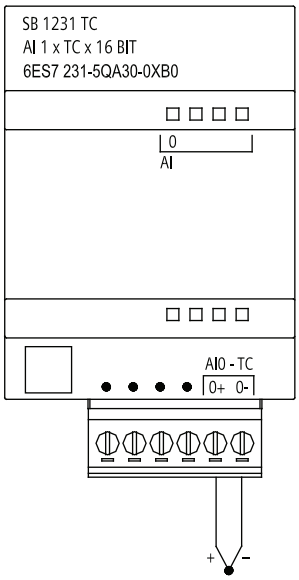
1 To maintain module resolution and accuracy when 400 Hz rejection is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

It is recommended for measuring thermocouples that a 100 ms integration time be used. The use of smaller integration times will increase the repeatability error of the temperature readings.

Note

After power is applied to the module, it performs internal calibration for the analog to digital converter. During this time, the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time.

Table A- 140 Wiring diagram for the analog input thermocouple SB

SB 1231 AI 1 x 16 bit thermocouple (6ES7 231-5QA30-0XB0)


A.9.5 RTD SBs

A.9.5.1 SB 1231 1 analog RTD input specifications

Note

To use this SB, your CPU firmware must be V2.0 or higher.

Table A- 141 General specifications

Technical data	SB 1231 AI 1 x 16 bit RTD
Order number	6ES7 231-5PA30-0XB0
Dimensions W x H x D (mm)	38 x 62 x 21 mm
Weight	35 grams
Power dissipation	0.7 W
Current consumption (SM Bus)	5 mA
Current consumption (24 VDC)	25 mA

Table A- 142 Analog inputs

Technical data		SB 1231 AI 1 x 16 bit RTD
Number of inputs		1
Type		Module referenced RTD and Ohms
Range • Nominal range (data word) • Overrange/underrange (data word) • Overflow/underflow (data word)		See Selection tables (Page 740).
Resolution	Temperature	0.1° C/ 0.1° F
	Voltage	15 bits plus sign
Maximum withstand voltage		±35 V
Noise rejection		85 dB (10 Hz, 50 Hz, 60 Hz, 400 Hz)
Common mode rejection		> 120 dB
Impedance		≥ 10 MΩ
Accuracy (25°C / 0 to 55°C)		See Selection tables (Page 740).
Repeatability		±0.05% FS
Maximum sensor dissipation		0.5 m W
Measuring principle		Integrating
Module update time		See Selection table (Page 740).
Isolation (field side to logic)		500 VAC
Cable length (meters)		100 m to sensor max.
Wire resistance		20 Ω, 2.7 for 10 Ω RTD max.

Table A- 143 Diagnostics

Technical data	SB 1231 AI 1 x 16 bit RTD
Overflow/underflow[1, 2]	Yes
Wire break [3]	Yes

1 The overflow and underflow diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

2 For resistance ranges underflow detection is never enabled.

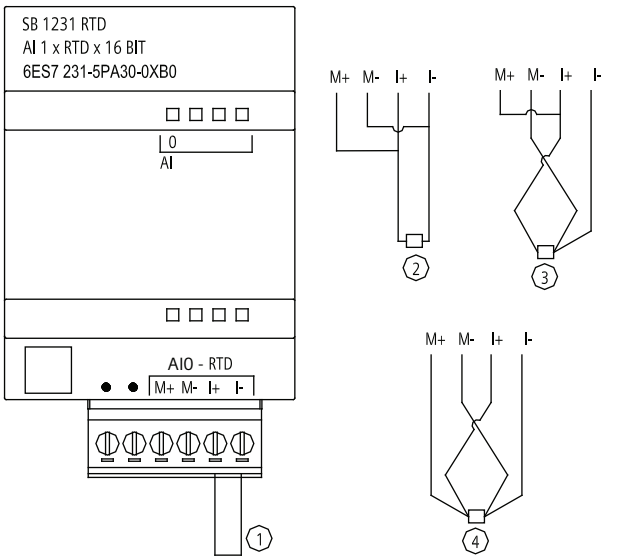
3 When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The SM 1231 RTD analog signal board measures the value of resistance connected to the signal board inputs. The measurement type can be selected as either "Resistor" or "Thermal resistor".

- "Resistor": The nominal range full scale value will be decimal 27648.
- "Thermal resistor": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).

The SB 1231 RTD signal board supports measurements with 2-wire, 3-wire and 4-wire connections to the sensor resistor.

Table A- 144 Wiring diagram for SB 1231 AI 1 x 16 bit RTD

SB 1213 AI 1 x 16 bit RTD (6ES7 231-5PA30-0XB0)


① Loop-back unused RTD input
② 2-wire RTD
③ 3-wire RTD
④ 4-wire RTD

A.9.5.2 Selection tables for the SB 1231 RTD

Table A- 145 Ranges and accuracy for the different sensors supported by the RTD modules

Temperature coefficient	RTD type	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range accuracy @ 25°C	Normal range accuracy 0°C to 55°C
Pt 0.003850	Pt 10	-243.0°C	-200.0°C	850.0°C	1000.0°C	±1.0°C	±2.0°C
ITS90 DIN EN 60751	Pt 50	-243.0°C	-200.0°C	850.0°C	1000.0°C	±0.5°C	±1.0°C
	Pt 100						
	Pt 200						
	Pt 500						
	Pt 1000						
Pt 0.003902 Pt 0.003916 Pt 0.003920	Pt 100	-243.0°C	-200.0°C	850.0°C	1000.0°C	± 0.5°C	±1.0°C
	Pt 200						
	Pt 500						
	Pt 1000						

Temperature coefficient	RTD type	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range accuracy @ 25°C	Normal range accuracy 0°C to 55°C
Pt 0.003910	Pt 10	-273.2°C	-240.0°C	1100.0°C	1295°C	±1.0°C	±2.0°C
	Pt 50	-273.2°C	-240.0°C	1100.0°C	1295°C	±0.8°C	±1.6°C
	Pt 100						
	Pt 500						
Ni 0.006720 Ni 0.006180	Ni 100	-105.0°C	-60.0°C	250.0°C	295.0°C	±0.5°C	±1.0°C
	Ni 120						
	Ni 200						
	Ni 500						
	Ni 1000						
LG-Ni 0.005000	LG-Ni 1000	-105.0°C	-60.0°C	250.0°C	295.0°C	±0.5°C	±1.0°C
Ni 0.006170	Ni 100	-105.0°C	-60.0°C	180.0°C	212.4°C	±0.5°C	±1.0°C
Cu 0.004270	Cu 10	-240.0°C	-200.0°C	260.0°C	312.0°C	±1.0°C	±2.0°C
Cu 0.004260	Cu 10	-60.0°C	-50.0°C	200.0°C	240.0°C	±1.0°C	±2.0°C
	Cu 50	-60.0°C	-50.0°C	200.0°C	240.0°C	±0.6°C	±1.2°C
	Cu 100						
Cu 0.004280	Cu 10	-240.0°C	-200.0°C	200.0°C	240.0°C	±1.0°C	±2.0°C
	Cu 50	-240.0°C	-200.0°C	200.0°C	240.0°C	±0.7°C	±1.4°C
	Cu 100						

1 RTD values below the under-range minimum value are reported as -32768.

2 RTD values above the over-range minimum value are reported as -32768.

Table A- 146 Resistance

Range	Under range minimum	Nominal range low limit	Nominal range high limit	Over range maximum[1]	Normal range accuracy @ 25°C	Normal range accuracy 0°C to 55°C
150 Ω	n/a	0 (0 Ω)	27648 (150 Ω)	176.383 Ω	±0.05%	±0.1%
300 Ω	n/a	0 (0 Ω)	27648 (300 Ω)	352.767 Ω	±0.05%	±0.1%
600 Ω	n/a	0 (0 Ω)	27648 (600 Ω)	705.534 Ω	±0.05%	±0.1%

1 RTD values above the over-range minimum value are reported as -32768.

Note

The module reports 32767 on any activated channel with no sensor connected. If open wire detection is also enabled, the module flashes the appropriate red LEDs.

When 500 Ω and 1000 Ω RTD ranges are used with other lower value resistors, the error may increase to two times the specified error.

Best accuracy will be achieved for the 10 Ω RTD ranges if 4 wire connections are used.

The resistance of the connection wires in 2 wire mode will cause an error in the sensor reading and therefore accuracy is not guaranteed.

Table A- 147 Noise reduction and update times for the RTD modules

Rejection frequency selection	Integration time	4-/2-wire, 1-channel module Update time (seconds)	3-wire, 1-channel module Update time (seconds)
400 Hz (2.5 ms)	10 ms [1]	0.036	0.071
60 Hz (16.6 ms)	16.67 ms	0.056	0.111
50 Hz (20 ms)	20 ms	0.066	1.086
10 Hz (100 ms)	100 ms	0.306	0.611

[1] To maintain module resolution and accuracy when the 400 Hz filter is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

NOTICE

After power is applied, the module performs internal calibration for the analog-to-digital converter. During this time the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time. Because the configuration of the module can vary the length of the initialization time, you should verify the behavior or the module in your configuration. If required, you can include logic in your user program to accommodate the initialization time of the module.

A.10 Communication interfaces

A.10.1 PROFIBUS

Note

S7-1200 PROFIBUS CMs and the GPRS CP are not approved for Maritime applications

The following modules do not have Maritime approval:

- CM 1242-5 PROFIBUS Slave module
- CM 1243-5 PROFIBUS Master module
- CP 1242-7 GPRS module

Note

To use these modules, your CPU firmware must be V2.0 or higher.

A.10.1.1 CM 1242-5

Table A- 148 Technical specifications of the CM 1242-5

Technical specifications	
Order number	6GK7 242-5DX30-0XE0
Interfaces	
Connection to PROFIBUS	9-pin D-sub female connector
Maximum current consumption on the PROFIBUS interface when network components are connected (for example optical network components)	15 mA at 5 V (only for bus termination) *)
Permitted ambient conditions	
Ambient temperature	
• during storage	• -40 °C to 70 °C
• during transportation	• -40 °C to 70 °C
• during operation with a vertical installation (DIN rail horizontal)	• 0 °C to 55 °C
• during operation with a horizontal installation (DIN rail vertical)	• 0 °C to 45 °C
Relative humidity at 25 °C during operation, without condensation, maximum	95 %
Degree of protection	IP20
Power supply, current consumption and power loss	

Technical specifications	
Type of power supply	DC
Power supply from the backplane bus	5 V
Current consumption (typical)	150 mA
Effective power loss (typical)	0.75 W
Dimensions and weights	
• Width	• 30 mm
• Height	• 100 mm
• Depth	• 75 mm
Weight	
• Net weight	• 115 g
• Weight including packaging	• 152 g

*)The current load of an external consumer connected between VP (pin 6) and DGND (pin 5) must not exceed a maximum of 15 mA (short-circuit proof) for bus termination.

PROFIBUS interface

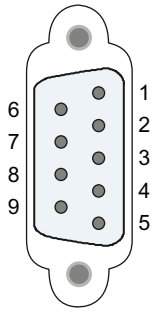


Table A- 149 Pinout of the D-sub socket

Pin	Description	Pin	Description
1	- not used -	6	P5V2: +5V power supply
2	- not used -	7	- not used -
3	RxD/TxD-P: Data line B	8	RxD/TxD-N: Data line A
4	RTS	9	- not used -
5	M5V2: Data reference potential (ground DGND)	Housing	Ground connector

A.10.1.2 CM 1243-5

Table A- 150 Technical specifications of the CM 1243-5

Technical specifications	
Order number	6GK7 243-5DX30-0XE0
Interfaces	
Connection to PROFIBUS	9-pin D-sub female connector
Maximum current consumption on the PROFIBUS interface when network components are connected (for example optical network components)	15 mA at 5 V (only for bus termination) *)
Permitted ambient conditions	

Technical specifications	
Ambient temperature	
• during storage	• -40 °C to 70 °C
• during transportation	• -40 °C to 70 °C
• during operation with a vertical installation (DIN rail horizontal)	• 0 °C to 55 °C
• during operation with a horizontal installation (DIN rail vertical)	• 0 °C to 45 °C
Relative humidity at 25 °C during operation, without condensation, maximum	95 %
Degree of protection	IP20
Power supply, current consumption and power loss	
Type of power supply	DC
Power supply / external	24 V
• minimum	• 19.2 V
• maximum	• 28.8 V
Current consumption (typical)	
• from 24 V DC	• 100 mA
• from the S7-1200 backplane bus	• 0 mA
Effective power loss (typical)	
• from 24 V DC	• 2.4 W
• from the S7-1200 backplane bus	• 0 W
Power supply 24 VDC / external	
• Min. cable cross section	• min.: 0.14 mm^2 (AWG 25)
• Max. cable cross section	• max.: 1.5 mm^2 (AWG 15)
• Tightening torque of the screw terminals	• 0.45 Nm (4 lb-in)
Dimensions and weights	
• Width	• 30 mm
• Height	• 100 mm
• Depth	• 75 mm
Weight	
• Net weight	• 134 g
• Weight including packaging	• 171 g

*)The current load of an external consumer connected between VP (pin 6) and DGND (pin 5) must not exceed a maximum of 15 mA (short-circuit proof) for bus termination.

Note

The CM 1243-5 (PROFIBUS master module) must receive power from the 24 VDC sensor supply of the CPU.

PROFIBUS interface

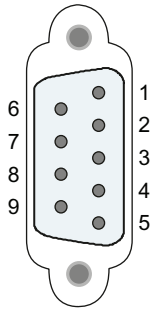


Table A- 151 Pinout of the D-sub socket

Pin	Description	Pin	Description
1	- not used -	6	VP: Power supply +5 V only for bus terminating resistors; not for supplying external devices
2	- not used -	7	- not used -
3	RxD/TxD-P: Data line B	8	RxD/TxD-N: Data line A
4	CNTR-P: RTS	9	- not used -
5	DGND: Ground for data signals and VP	Housing	Ground connector

PROFIBUS cable

NOTICE

Contacting the shield of the PROFIBUS cable

The shield of the PROFIBUS cable must be contacted.

To do this, strip the insulation from the end of the PROFIBUS cable and connect the shield to functional earth.

A.10.2 GPRS

Note

S7-1200 PROFIBUS CMs and the GPRS CP are not approved for Maritime applications

The following modules do not have Maritime approval:

- CM 1242-5 PROFIBUS Slave module
- CM 1243-5 PROFIBUS Master module
- CP 1242-7 GPRS module

Note

To use these modules, your CPU firmware must be V2.0 or higher.

A.10.2.1 CP 1242-7

Table A- 152 Technical specifications of the CP 1242-7

Technical specifications	
Order number	6GK7 242-7KX30-0XE0
Wireless interface	
Antenna connector	SMA socket
Nominal impedance	50 ohms
Wireless connection	
Maximum transmit power	• GSM 850, class 4: +33 dBm ±2dBm • GSM 900, class 4: +33 dBm ±2dBm • GSM 1800, class 1: +30 dBm ±2dBm • GSM 1900, class 1: +30 dBm ±2dBm
GPRS	Multislot class 10 device class B coding scheme 1...4 (GMSK)
SMS	Mode outgoing: MO service: point-to-point
Permitted ambient conditions	
Ambient temperature	
• during storage	• -40 °C to 70 °C
• during transportation	• -40 °C to 70 °C
• during operation with a vertical installation (DIN rail horizontal)	• 0 °C to 55 °C
• during operation with a horizontal installation (DIN rail vertical)	• 0 °C to 45 °C
Relative humidity at 25 °C during operation, without condensation, maximum	95 %
Degree of protection	IP20
Power supply, current consumption and power loss	
Type of power supply	DC
Power supply / external	24 V
• minimum	• 19.2 V
• maximum	• 28.8 V
Current consumption (typical)	
• from 24 V DC	• 100 mA
• from the S7-1200 backplane bus	• 0 mA
Effective power loss (typical)	
• from 24 V DC	• 2.4 W
• from the S7-1200 backplane bus	• 0 W

Technical specifications	
24 V DC power supply	
• Min. cable cross section	• min.: 0.14 mm² (AWG 25)
• Max. cable cross section	• max.: 1.5 mm² (AWG 15)
• Tightening torque of the screw terminals	• 0.45 Nm (4 lb-in)
Dimensions and weights	
• Width	• 30 mm
• Height	• 100 mm
• Depth	• 75 mm
Weight	
• Net weight	• 133 g
• Weight including packaging	• 170 g

Technical specifications of the ANT794-4MR GSM/GPRS antenna

ANT794-4MR	
Order number	6NH9860-1AA00
Mobile wireless networks	GSM/GPRS
Frequency ranges	• 824 to 960 MHz (GSM 850, 900) • 1 710 to 1 880 MHz (GSM 1 800) • 1 900 to 2 200 MHz (GSM / UMTS)
Characteristics	omnidirectional
Antenna gain	0 dB
Impedance	50 ohms
Standing wave ratio (SWR)	< 2,0
Max. power	20 W
Polarity	linear vertical
Connector	SMA
Length of antenna cable	5 m
External material	Hard PVC, UV-resistant
Degree of protection	IP20
Permitted ambient conditions	
• Operating temperature	• -40 °C through +70 °C
• Transport/storage temperature	• -40 °C through +70 °C
• Relative humidity	• 100 %
External material	Hard PVC, UV-resistant
Construction	Antenna with 5 m fixed cable and SMA male connector
Dimensions (D x H) in mm	25 x 193

ANT794-4MR	
Weight	
• Antenna incl. cable	• 310 g
• Fittings	• 54 g
Installation	With supplied bracket

Technical specifications of the flat antenna ANT794-3M

Order number	6NH9870-1AA00	
Mobile wireless networks	**GSM 900**	**GSM 1800/1900**
Frequency ranges	890 - 960 MHz	1710 - 1990 MHz
Standing wave ratio (VSWR)	≤ 2:1	≤ 1,5:1
Return loss (Tx)	≈ 10 dB	≈ 14 dB
Antenna gain	0 dB	
Impedance	50 ohms	
Max. power	10 W	
Antenna cable	HF cable RG 174 (fixed) with SMA male connector	
Cable length	1.2 m	
Degree of protection	IP64	
Permitted temperature range	-40°C to +75°C	
Flammability	UL 94 V2	
External material	ABS Polylac PA-765, light gray (RAL 7035)	
Dimensions (W x L x H) in mm	70.5 x 146.5 x 20.5	
Weight	130 g	

A.10.3 CM 1243-2 AS-i Master

A.10.3.1 Technical data for the AS-i master CM 1243-2

Table A- 153 Technical data for the AS-i master CM 1243-2

Technical data	
Order number	3RK7243-2AA30-0XB0
Interfaces	
Maximum current consumption	
From the SIMATIC backplane bus	Max. 250 mA, SIMATIC backplane bus supply voltage 5 V DC
From the AS-i cable	Max. 100 mA
Pin assignment	See section Electrical connections of the AS-i master CM 1243-2 (Page 750)

Technical data	
Conductor cross-section	0.2 mm² (AWG 24) ... 3.3 mm² (AWG 12)
ASI connector tightening torque	0.56 Nm
Permissible ambient conditions	
Ambient temperature	
During storage	-40 °C ... 70 °C
During transport	-40 °C ... 70 °C
During the operating phase, with vertical installation (horizontal standard mounting rail)	0 °C ... 55 °C
During the operating phase, with horizontal installation (vertical standard mounting rail)	0 °C ... 45 °C
Relative humidity at 25 °C during operating phase, no condensation, maximum	95 %
Degree of protection	IP20
Power supply, current consumption, power loss	
Type of power supply	DC
Current consumption (typically)	
From the S7-1200 backplane bus	200 mA
Power loss (typically) From the S7-1200 backplane bus	2.4 W from AS-i 0.5 W
Dimensions and weights	
Width	30 mm
Height	100 mm
Depth	75 mm
Weight	
Net weight	122 g
Weight including packaging	159 g

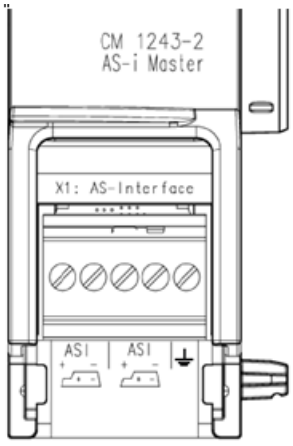
A.10.3.2 Electrical connections of the AS-i master CM 1243-2

Power supply of the AS-i master CM 1243-2

The AS-i master CM 1243-2 is supplied over the communications bus of the S7-1200. This means that a diagnostics message can still be sent to the S7-1200 following failure of the AS-i supply voltage. The connection to the communications bus is on the right-hand side of the AS-i master CM 1243-2.

AS-Interface terminals

The removable terminal for connecting the AS-i cable is located behind the lower cover on the front of the AS-i master CM 1243-2.



If the AS-i shaped cable is used, you can recognize the correct polarity of the cable by means of the symbol .

Information on how to remove and re-install the terminal block can be found in the system manual "SIMATIC S7-1200 Programmable Controller" (Order No.: 6ES7298-8FA30-8AH0).

NOTICE
Maximum current carrying capacity of the terminal contacts The current carrying capacity of the connection contacts is max. 8 A. If this value is exceeded on the AS-i cable, the AS-i master CM 1243-2 must not be "looped in" to the AS-i cable, but must instead be connected via a spur line (only one connection pair assigned on the AS-i master CM 1243-2). You will find additional information on connecting the AS-i cable in the section "Installation, connection and commissioning of the modules" in the manual “AS-i Master CM 1243-2 and AS-i data decoupling unit DCM 1271 for SIMATIC S7-1200”.

Terminal assignment:

Label	Meaning
ASI+	AS-i connection – positive polarity
ASI–	AS-i connection – negative polarity
⏚	Functional ground

Additional information on the electrical connections

Read section Auto hotspot for instructions on connecting the electrical connections.

You can find technical details on the electrical connections in section Auto hotspot.

A.10.4 RS232, RS422, and RS485

A.10.4.1 CB 1241 RS485 Specifications

Note

To use this CB, your CPU firmware must be V2.0 or higher.

Table A- 154 General specifications

Technical data	CB 1241 RS485
Order number	6ES7 241-1CH30-1XB0
Dimensions	38 x 62 x 21
Weight	40 grams

Table A- 155 Transmitter and receiver

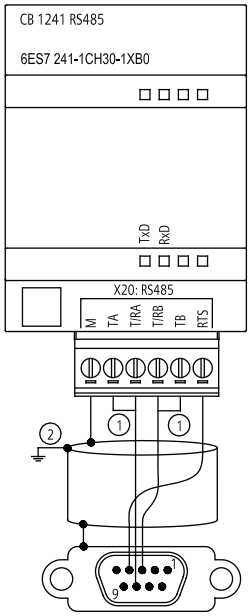
Technical data	CB 1241 RS485
Type	RS485 (2-wire half-duplex)
Common mode voltage range	-7 V to +12 V, 1 second, 3 VRMS continuous
Transmitter differential output voltage	2 V min. at R_L = 100 Ω 1.5 V min. at R_L = 54 Ω
Termination and bias	10K to +5 V on B, RS485 Pin 3 10K to GND on A, RS485 Pin 4
Optional termination	Short Pin TB to Pin T/RB, effective termination impedance is 127 Ω, connects to RS485 Pin 3 Short Pin TA to Pin T/RA, effective termination impedance is 127 Ω, connects to RS485 Pin 4
Receiver input impedance	5.4K Ω min. including termination
Receiver threshold/sensitivity	+/- 0.2 V min., 60 mV typical hysteresis
Isolation RS485 signal to chassis ground RS485 signal to CPU logic common	500 VAC, 1 minute
Cable length, shielded	1000 m max.
Baud rate	300 baud, 600 baud, 1.2 kbits, 2.4 kbits, 4.8 kbits, 9.6 kbits (default), 19.2 kbits, 38.4 kbits, 57.6 kbits, 76.8 kbits, 115.2 kbits,

Technical data	CB 1241 RS485
Parity	No parity (default), even, odd, Mark (parity bit always set to 1), Space (parity bit always set to 0)
Number of stop bits	1 (default), 2
Flow control	Not supported
Wait time	0 to 65535 ms

Table A- 156 Power supply

Technical data	CB 1241 RS485
Power loss (dissipation)	1.5 W
Current consumption (SM Bus), max.	50 mA
Current consumption (24 VDC) max.	80 mA

CB 1241 RS485 (6ES7 241-1CH30-1XB0)



① Connect "TA" and TB" as shown to terminate the network. (Terminate only the end devices on the RS485 network.)

② Use shielded twisted pair cable and connect the cable shield to ground.

You terminate only the two ends of the RS485 network. The devices in between the two end devices are not terminated or biased. See the section on "Biasing and terminating an RS485 network connector" (Page 530).

A.10.4.2 CM 1241 RS485 Specifications

Table A- 157 General specifications

Technical data	CM 1241 RS485
Order number	6ES7 241-1CH30-0XB0
Dimensions	30 x 100 x 75 mm
Weight	150 grams

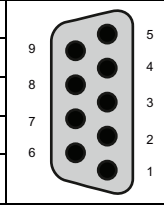
Table A- 158 Transmitter and receiver

Technical data	CM 1241 RS485
Type	RS485 (2-wire half-duplex)
Common mode voltage range	-7 V to +12 V, 1 second, 3 VRMS continuous
Transmitter differential output voltage	2 V min. at R_L = 100 Ω 1.5 V min. at R_L = 54 Ω
Termination and bias	10K Ω to +5 V on B, PROFIBUS Pin 3 10K Ω to GND on A, PROFIBUS Pin 8
Receiver input impedance	5.4K Ω min. including termination
Receiver threshold/sensitivity	+/- 0.2 V min., 60 mV typical hysteresis
Isolation RS485 signal to chassis ground RS485 signal to CPU logic common	500 VAC, 1 minute
Cable length, shielded	1000 m max.
Baud rate	300 baud, 600 baud, 1.2 kbits, 2.4 kbits, 4.8 kbits, 9.6 kbits (default), 19.2 kbits, 38.4 kbits, 57.6 kbits, 76.8 kbits, 115.2 kbits,
Parity	No parity (default), even, odd, Mark (parity bit always set to 1), Space (parity bit always set to 0)
Number of stop bits	1 (default), 2
Flow control	Not supported
Wait time	0 to 65535 ms

Table A- 159 Power supply

Technical data	CM 1241 RS485
Power loss (dissipation)	1.1 W
From +5 VDC	220 mA

Table A- 160 RS485 connector (female)

Pin	Description	Connector (female)	Pin	Description
1	Not connected		6 PWR	+5V with 100 ohm series resistor: Output
2	Not connected		7	Not connected
3 TxD+	Signal B (RxD/TxD+): Input/Output		8 TXD-	Signal A (RxD/TxD-): Input/Output
4 RTS [1]	Request to send (TTL level): Output		9	Not connected
5 GND	Logic or communication ground		SHELL	Chassis ground

[1] The RTS is a TTL level signal and can be used to control another half duplex device based on this signal. It is enabled when you transmit and is disabled all other times. Unlike with the CM 1241 RS232, there is no user control of this signal on the CM 1241 RS485. You cannot set it manually or cause it to be extended.

A.10.4.3 CM 1241 RS232 Specifications

Table A- 161 General specifications

Technical data	CM 1241 RS232
Order number	6ES7 241-1AH30-0XB0
Dimensions	30 x 100 x 75 mm
Weight	150 grams

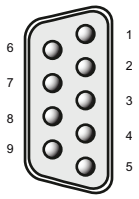
Table A- 162 Transmitter and receiver

Technical data	CM 1241 RS232
Type	RS232 (full-duplex)
Transmitter output voltage	+/- 5 V min. at R_L = 3K Ω
Transmit output voltage	+/- 15 VDC max.
Receiver input impedance	3 K Ω min.
Receiver threshold/sensitivity	0.8 V min. low, 2.4 max. high 0.5 V typical hysteresis
Receiver input voltage	+/- 30VDC max.
Isolation RS 232 signal to chassis ground RS 232 signal to CPU logic common	500 VAC, 1 minute
Cable length, shielded	10 m max.
Baud rate	300 baud, 600 baud, 1.2 kbits, 2.4 kbits, 4.8 kbits, 9.6 kbits (default), 19.2 kbits, 38.4 kbits, 57.6 kbits, 76.8 kbits, 115.2 kbits,
Parity	No parity (default), even, odd, Mark (parity bit always set to 1), Space (parity bit always set to 0)
Number of stop bits	1 (default), 2
Flow control	Hardware, software
Wait time	0 to 65535 ms

Table A- 163 Power supply

Technical data	CM 1241 RS232
Power loss (dissipation)	1.1 W
From +5 VDC	220 mA

Table A- 164 RS232 connector (male)

Pin	Description	Connector (male)	Pin	Description
1 DCD	Data carrier detect: Input		6 DSR	Data set ready: Input
2 RxD	Received data from DCE: Input		7 RTS	Request to send: Output
3 TxD	Transmitted data to DCE: Output		8 CTS	Clear to send: Input
4 DTR	Data terminal ready: Output		9 RI	Ring indicator (not used)
5 GND	Logic ground		SHELL	Chassis ground

A.10.4.4 CM 1241 RS422/485 Specifications

CM 1241 RS422/485 Specifications

Table A- 165 General specifications

Technical data	CM 1241 RS422/485
Order number	6ES7 241-1CH31-0XB0
Dimensions	30 x 100 x 75 mm
Weight	155 grams

Table A- 166 Transmitter and receiver

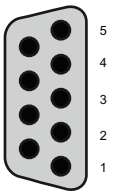
Technical data	CM 1241 RS422/485
Type	RS422 or RS485, 9-pin sub D female connector
Common mode voltage range	-7 V to +12 V, 1 second, 3 VRMS continuous
Transmitter differential output voltage	2 V min. at R_L = 100 Ω 1.5 V min. at R_L = 54 Ω
Termination and bias	10K Ω to +5 V on B, PROFIBUS Pin 3 10K Ω to GND on A, PROFIBUS Pin 8 Internal bias options provided, or no internal bias. In all cases, external termination is required (*see Chapter 12, Communcations Protocols, page xxx)*
Receiver input impedance	5.4K Ω min. including termination

Technical data	CM 1241 RS422/485
Receiver threshold/sensitivity	+/- 0.2 V min., 60 mV typical hysteresis
Isolation RS485 signal to chassis ground RS485 signal to CPU logic common	500 VAC, 1 minute
Cable length, shielded	1000 m max. (baud rate dependent)
Baud rate	300 baud, 600 baud, 1.2 kbits, 2.4 kbits, 4.8 kbits, 9.6 kbits (default), 19.2 kbits, 38.4 kbits, 57.6 kbits, 76.8 kbits, 115.2 kbits,
Parity	No parity (default), even, odd, Mark (parity bit always set to 1), Space (parity bit always set to 0)
Number of stop bits	1 (default), 2
Flow control	XON/XOFF supported for the RS422 mode
Wait time	0 to 65535 ms

Table A- 167 Power supply

Technical data	CM 1241 RS422/485
Power loss (dissipation)	1.2 W
From +5 VDC	240 mA

Table A- 168 RS485 or RS422 connector (female)

Pin	Description	Connector (female)	Pin	Description
1	Logic or communication ground		6 PWR	+5V with 100 ohm series resistor: Output
2 TxD+ [1]	Connected for RS422 Not used for RS485: Output		7	Not connected
3 TxD+	Signal B (RxD/TxD+): Input/Output		8 TXD-	Signal A (RxD/TxD-): Input/Output
4 RTS [2]	Request to send (TTL level) Output		9 TXD- [1]	Connected for RS422 Not used for RS485: Output
5 GND	Logic or communication ground		SHELL	Chassis ground

[1] Pins 2 and 9 are only used as transmit signals for RS422.

[2] The RTS is a TTL level signal and can be used to control another half duplex device based on this signal. It is active when you transmit and is inactive all other times.

A.11 TeleService (TS Adapter and TS Adapter modular)

The following manuals contain the technical specification for the TS Adapter IE Basic and the TS Adapter modular:

- Industrial Software Engineering Tools
  Modular TS Adapter
- Industrial Software Engineering Tools
  TS Adapter IE Basic

A.12 SIMATIC memory cards

Order Number	Capacity
6ES7 954-8LF01-0AA0	24 MB
6ES7 954-8LE01-0AA0	12 MB
6ES7 954-8LB01-0AA0	2 MB

A.13 Input simulators

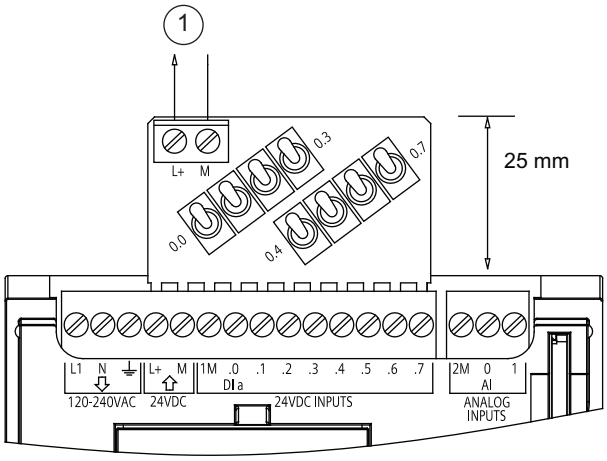
Table A- 169 General specifications

Technical data	8 Position Simulator	14 Position Simulator
Order number	6ES7 274-1XF30-0XA0	6ES7 274-1XH30-0XA0
Dimensions W x H x D (mm)	43 x 35 x 23	67 x 35 x 23
Weight	20 grams	30 grams
Points	8	14
Used with CPU	CPU 1211C, CPU 1212C	CPU 1214C

⚠ WARNING

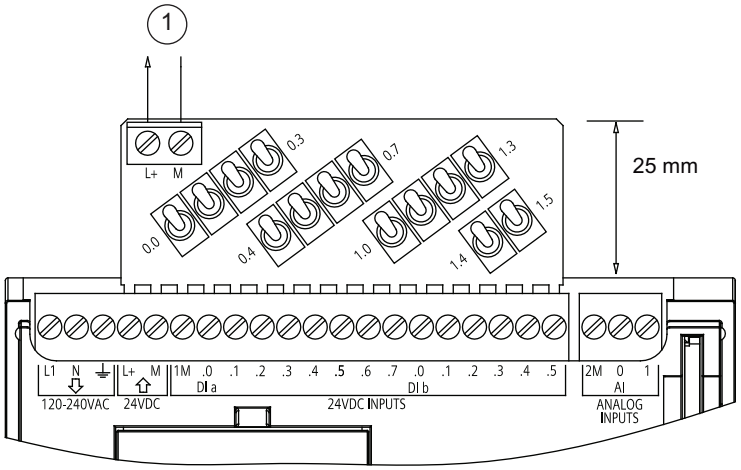
These input simulators are not approved for use in Class I DIV 2 or Class I Zone 2 hazardous locations. The switches present a potential spark hazard/explosion hazard if used in a Class I DIV 2 or Class I Zone 2 location.

8 Position Simulator (6ES7 274-1XF30-0XA0)



① 24 VDC sensor power out

14 Position Simulator (6ES7 274-1XF30-0XA0)

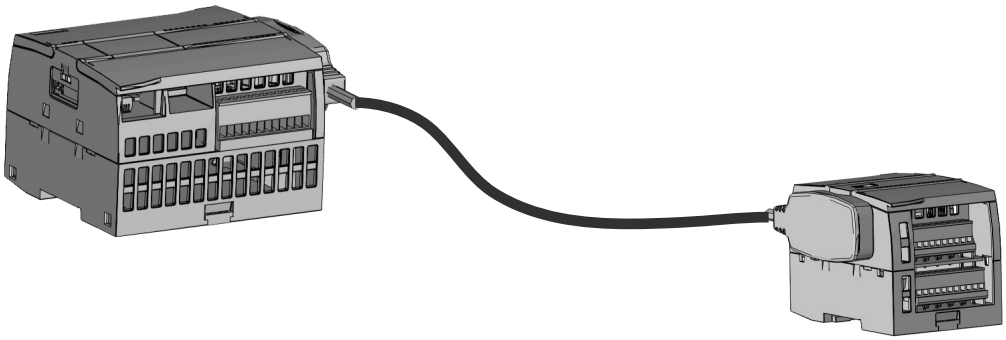


① 24 VDC sensor power out

A.14 I/O expansion cable

Technical Data	
Order number	6ES7 290-6AA30-0XA0
Cable length	2 m
Weight	200 g

Refer to the installation section (Page 51) for information about installing and removing the S7-1200 expansion cable.



A.15 Companion products

A.15.1 PM 1207 power module

The PM 1207 is a power supply module for the SIMATIC S7-1200. It provides the following features:

- Input 120/230 VAC, output 24 VDC/2.5A
- Order number 6ESP 332-1SH71

For more information about this product and for the product documentation, refer to the customer support web site (http://www.siemens.com/automation/support-request).

A.15.2 CSM 1277 compact switch module

The CSM1277 is an Industrial Ethernet compact switch module. It can be used to multiply the Ethernet interface of the S7-1200 to allow simultaneous communication with operator panels, programming devices, or other controllers. It provides the following features:

- 4 x RJ45 sockets for connecting to Industrial Ethernet
- 3 pole plug in terminal strip for connection of the external 24 VDC supply on top
- LEDs for diagnostics and status display of Industrial Ethernet ports
- Order number 6GK7 277-1AA00-0AA0

For more information about this product and for the product documentation, refer to the customer support web site (http://www.siemens.com/automation/support-request).

Calculating a power budget

B

The CPU has an internal power supply that provides power for the CPU itself, for any expansion modules, and for other 24 VDC user power requirements.

There are four types of expansion modules:

- Signal modules (SM) are installed on the right-side of the CPU. Each CPU allows a maximum number of signal modules possible without regard to the power budget.
  - CPU 1214 allows 8 signal modules
  - CPU 1212 allows 2 signal modules
  - CPU 1211 allows no signal modules
- Communication modules (CM) are installed on the left-side of the CPU. A maximum of 3 communication modules is allowed for any CPU without regard to the power budget.
- Signal boards (SB) are installed on top of the CPU. A maximum of 1 signal board or communication board is allowed for any CPU.
- Communication boards (CB) are installed on top of the CPU. A maximum of 1 signal board or communication board is allowed for any CPU.

Use the following information as a guide for determining how much power (or current) the CPU can provide for your configuration.

Each CPU supplies both 5 VDC and 24 VDC power:

- The CPU provides 5 VDC power for the expansion modules when an expansion module is connected. If the 5 VDC power requirements for expansion modules exceed the power budget of the CPU, you must remove expansion modules until the requirement is within the power budget.
- Each CPU has a 24 VDC sensor supply that can supply 24 VDC for local input points or for relay coils on the expansion modules. If the power requirement for 24 VDC exceeds the power budget of the CPU, you can add an external 24 VDC power supply to provide 24 VDC to the expansion modules. You must manually connect the 24 VDC supply to the input points or relay coils.

⚠ WARNING

Connecting an external 24 VDC power supply in parallel with the DC sensor supply can result in a conflict between the two supplies as each seeks to establish its own preferred output voltage level.

The result of this conflict can be shortened lifetime or immediate failure of one or both power supplies, with consequent unpredictable operation of the PLC system. Unpredictable operation could result in death, severe personal injury and/or property damage.

The DC sensor supply on the CPU and any external power supply should provide power to different points. A single connection of the commons is allowed.

Some of the 24V power input ports in the PLC system are interconnected, with a logic common circuit connecting multiple M terminals. The CPU 24V power supply input, the SM relay coil power input, and a non-isolated analog power supply input are examples of circuits that are interconnected when designated as not isolated in the data sheets. All non-isolated M terminals must connect to the same external reference potential.

⚠ WARNING

Connecting non-isolated M terminals to different reference potentials will cause unintended current flows that may cause damage or unpredictable operation in the PLC and connected equipment.

Such damage or unpredictable operation could result in death, severe personal injury and/or property damage.

Always be sure that all non-isolated M terminals in a PLC system are connected to the same reference potential.

Information about the power budgets of the CPUs and the power requirements of the signal modules is provided in the technical specifications (Page 657).

Note

Exceeding the power budget of the CPU may result in not being able to connect the maximum number of modules allowed for your CPU.

Sample power budget

The following example shows a sample calculation of the power requirements for a configuration that includes one CPU 1214C AC/DC/Relay, one SB 1223 2 x 24 VDC Input/ 2 x 24 VDC Output, one CM 1241, three SM 1223 8 DC In/8 Relay Out, and one SM 1221 8 DC In. This example has a total of 48 inputs and 36 outputs.

Note

The CPU has already allocated the power required to drive the internal relay coils. You do not need to include the internal relay coil power requirements in a power budget calculation.

The CPU in this example provides sufficient 5 VDC current for the SMs, but does not provide enough 24 VDC current from the sensor supply for all of the inputs and expansion relay coils. The I/O requires 456 mA and the CPU provides only 400 mA. This installation requires an additional source of at least 56 mA at 24 VDC power to operate all the included 24 VDC inputs and outputs.

Table B- 1 Sample power budget

CPU power budget	5 VDC	24 VDC
CPU 1214C AC/DC/Relay	1600 mA	400 mA
Minus		
System requirements	**5 VDC**	**24 VDC**
CPU 1214C, 14 inputs	-	14 * 4 mA = 56 mA
1 SB 1223 2 x 24 VDC Input/ 2 x 24 VDC Output	50 mA	2 * 4 mA = 8 mA
1 CM 1241 RS485, 5 V power	220 mA	
3 SM 1223, 5 V power	3 * 145 mA = 435 mA	-
1 SM 1221, 5 V power	1 * 105 mA = 105 mA	-
3 SM 1223, 8 inputs each	-	3 * 8 * 4 mA = 96 mA
3 SM 1223, 8 relay coils each	-	3 * 8 * 11 mA = 264 mA
1 SM 1221, 8 inputs each	-	8 * 4 mA = 32 mA
Total requirements	810 mA	456 mA
Equals		
Current balance	**5 VDC**	**24 VDC**
Current balance total	790 mA	(56 mA)

Form for calculating your power budget

Use the following table to determine how much power (or current) the S7-1200 CPU can provide for your configuration. Refer to the technical specifications (Page 657) for the power budgets of your CPU model and the power requirements of your signal modules.

Table B- 2 Calculations for a power budget

CPU power budget	5 VDC	24 VDC
Minus		
System requirements	**5 VDC**	**24 VDC**
Total requirements		
Equals		
Current balance	**5 VDC**	**24 VDC**
Current balance total		

Order numbers

C

C.1 CPU modules

Table C- 1 S7-1200 CPUs

CPU models		Order Number
CPU 1211C	CPU 1211C DC/DC/DC	6ES7 211-1AD30-0XB0
	CPU 1211C AC/DC/Relay	6ES7 211-1BD30-0XB0
	CPU 1211C DC/DC/Relay	6ES7 211-1HD30-0XB0
CPU 1212C	CPU 1212C DC/DC/DC	6ES7 212-1AD30-0XB0
	CPU 1212C AC/DC/Relay	6ES7 212-1BD30-0XB0
	CPU 1212C DC/DC/Relay	6ES7 212-1HD30-0XB0
CPU 1214C	CPU 1214C DC/DC/DC	6ES7 214-1AE30-0XB0
	CPU 1214C AC/DC/Relay	6ES7 214-1BE30-0XB0
	CPU 1214C DC/DC/Relay	6ES7 214-1HE30-0XB0

C.2 Signal modules (SMs) and signal boards (SBs)

Table C- 2 Signal boards (SBs)

Signal boards		Order Number
Digital input	SB 1221 200 KHz 4 x 24 VDC Input (Source),	6ES7 221-3BD30-0XB0
	SB 1221 200 KHz 4 x 5 VDC Input (Source)	6ES7 221-3AD30-0XB0
Digital output	SB 1222 200 KHz 4 x 24 VDC Output (Sink/Source)	6ES7 222-1BD30-0XB0
	SB 1222 200 KHz 4 x 5 VDC Output (Sink/Source)	6ES7 222-1AD30-0XB0
Digital input / output	SB 1223 2 x 24 VDC Input (Sink) / 2 x 24 VDC Output (Source)	6ES7 223-0BD30-0XB0
	SB 1223 200 KHz 2 x 24 VDC Input (Source) / 2 x 24 VDC Output (Sink/Source)	6ES7 223-3BD30-0XB0
	SB 1223 200 KHz 2 x 5 VDC Input (Source) / 2 x 5 VDC Output (Sink/Source)	6ES7 223-3AD30-0XB0
Analog	SB 1232 1 Analog Output	6ES7 232-4HA30-0XB0
	SB 1231 1 Analog Input	6ES7 231-4HA30-0XB0
	SB 1231 1 Analog Input Thermocouple	6ES7 231-5QA30-0XB0
	SB 1231 1 Analog Input RTD	6ES7 231-5PA30-0XB0

Table C- 3 Signal modules (SMs)

Signal modules		Order Number
Digital input	SM 1221 8 x 24 VDC Input (Sink/Source)	6ES7 221-1BF30-0XB0
	SM 1221 16 x 24 VDC Input (Sink/Source)	6ES7 221-1BH30-0XB0
Digital output	SM 1222 8 x 24 VDC Output (Source)	6ES7 222-1BF30-0XB0
	SM 1222 16 x 24 VDC Output (Source)	6ES7 222-1BH30-0XB0
	SM 1222 8 x Relay Output	6ES7 222-1HF30-0XB0
	SM 1222 8 x Relay Output (Changeover)	6ES7 222-1XF30-0XB0
	SM 1222 16 x Relay Output	6ES7 222-1HH30-0XB0
Digital input / output	SM 1223 8 x 24 VDC Input (Sink/Source) / 8 x 24 VDC Output (Source)	6ES7 223-1BH30-0XB0
	SM 1223 16 x 24 VDC Input (Sink/Source) / 16 x 24 VDC Output (Source)	6ES7 223-1BL30-0XB0
	SM 1223 8 x 24 VDC Input (Sink/Source) / 8 x Relay Output	6ES7 223-1PH30-0XB0
	SM 1223 16 x 24 VDC Input (Sink/Source) / 16 x Relay Output	6ES7 223-1PL30-0XB0
	SM 1223 8 x 120/230 VAC Input (Sink/Source) / 8 x Relay Outputs	6ES7 223-1QH30-0XB0
Analog input	SM 1231 4 x Analog Input	6ES7 231-4HD30-0XB0
	SM 1231 8 x Analog Input	6ES7 231-4HF30-0XB0
Analog output	SM 1232 2 x Analog Output	6ES7 232-4HB30-0XB0
	SM 1232 4 x Analog Output	6ES7 232-4HD30-0XB0
Analog input / output	SM 1234 4 x Analog Input / 2 x Analog Output	6ES7 234-4HE30-0XB0
RTD and thermocouple	SM 1231 TC 4 x 16 bit	6ES7 231-5QD30-0XB0
	SM 1231 TC 8 x 16 bit	6ES7 231-5QF30-0XB0
	SM 1231 RTD 4 x 16 bit	6ES7 231-5PD30-0XB0
	SM 1231 RTD 8 x 16 bit	6ES7 231-5PF30-0XB0

C.3 Communication

Table C- 4 Communication module (CM)

Communication module (CM)			Order Number
RS232 and RS485	CM 1241 RS232	RS232	6ES7 241-1AH30-0XB0
	CM 1241 RS485	RS485	6ES7 241-1CH30-0XB0
	CM 1241 RS422/485	RS422/485	6ES7 241-1CH31-0XB0
PROFIBUS	CM 1243-5	PROFIBUS Master	6GK7 243-5DX30-0XE0
	CM 1242-5	PROFIBUS Slave	6GK7 242-5DX30-0XE0
AS-i Master	CM 1243-2	AS-i Master	3RK7 243-2AA30-0XB0

Table C- 5 Communication board (CB)

Communication board (CB)			Order Number
RS485	CB 1241 RS485	RS485	6ES7 241-1CH30-1XB0

Table C- 6 Communication Processor (CP)

Communication processor (CP)		Order Number
CP 1242-7	GPRS	6GK7 242-7KX30-0XE0

Table C- 7 TeleService

TS Adapter	Order Number
TS Adapter IE Basic	6ES7 972-0EB00-0XA0
TS Module GSM	6GK7 972-0MG00-0XA0
TS Module RS232	6ES7 792-0MS00-0XA0
TS Module Modem	6ES7 972-0MM00-0XA0
TS Module ISDN	6ES7 972-0MD00-0XA0

Table C- 8 Accessories

Accessory			Order Number
Antenna	ANT794-4MR	GSM/GPRS antenna	6NH9 860-1AA00
	ANT794-3M	Flat antenna	6NH9 870-1AA00

Table C- 9 Connectors

Type of Connector		Order Number
RS485	35-degree cable output, screw-terminal connection	6ES7 972-0BA42-0XA0
	35-degree cable output, FastConnect connection	6ES7 972-0BA60-0XA0

C.4 Other modules

Table C- 10 Companion products

Item		Order Number
Power supply module	PM 1207 power supply	6EP1 332-1SH71
Ethernet switch	CSM 1277 Ethernet switch - 4 ports	6GK7 277-1AA10-0AA0

C.5 Memory cards

Table C- 11 Memory cards

SIMATIC memory cards	Order Number
SIMATIC MC 2 MB	6ES7 954-8LB01-0AA0
SIMATIC MC 12 MB	6ES7 954-8LE01-0AA0
SIMATIC MC 24 MB	6ES7 954-8LF01-0AA0

C.6 Basic HMI devices

Table C- 12 HMI devices

HMI Basic Panels	Order Number
KTP400 Basic (Mono, PN)	6AV6 647-0AA11-3AX0
KTP600 Basic (Mono, PN)	6AV6 647-0AB11-3AX0
KTP600 Basic (Color, PN)	6AV6 647-0AD11-3AX0
KTP1000 Basic (Color, PN)	6AV6 647-0AF11-3AX0
TP1500 Basic (Color, PN)	6AV6 647-0AG11-3AX0

C.7 Spare parts and other hardware

Table C- 13 Expansion cables, simulators and connector blocks

Item			Order Number
I/O expansion cable	I/O Expansion cable, 2 m		6ES7 290-6AA30-0XA0
I/O simulator	Simulator (1214C/1211C - 8 position)		6ES7 274-1XF30-0XA0
	Simulator (1214C - 14 position)		6ES7 274-1XH30-0XA0
Spare door kit	CPU 1211/1212		6ES7 291-1AA30-0XA0
	CPU 1214		6ES7 291-1AB30-0XA0
	Signal module, 45 mm		6ES7 291-1BA30-0XA0
	Signal module, 70 mm		6ES7 291-1BB30-0XA0
	Communication module		6ES7 291-1CC30-0XA0
Connector block	Tin	7 terminal, 4/pk	6ES7 292-1AG30-0XA0
		8 terminal, 4/pk	6ES7 292-1AH30-0XA0
		11 terminal, 4/pk	6ES7 292-1AL30-0XA0
		12 terminal, 4/pk	6ES7 292-1AM30-0XA0
		14 terminal, 4/pk	6ES7 292-1AP30-0XA0
		20 terminal, 4/pk	6ES7 292-1AV30-0XA0
	Gold	3 terminal, 4/pk	6ES7 292-1BC30-0XA0

Item			Order Number
		6 terminal, 4/pk	6ES7 292-1BF30-0XA0
		7 terminal, 4/pk	6ES7 292-1BG30-0XA0
		11 terminal, 4/pk	6ES7 292-1BL30-0XA0
Strain relief	Strain Relief, CPU1200, Ethernet (4/pk)		6ES7 290-3AA30-0XA0

C.8 Programming software

Table C- 14 Programming software

SIMATIC software		Order Number
Programming software	STEP 7 Basic V11	6ES7 822-0AA01-0YA0
	STEP 7 Professional V11	6ES7 822-1AA01-0YA5
Visualization software	WinCC Basic V11	6AV2100-0AA01-0AA0
	WinCC Comfort V11	6AV2101-0AA01-0AA5
	WinCC Advanced V11	6AV2102-0AA01-0AA5
	WinCC Professional 512 PowerTags V11	6AV2103-0DA01-0AA5
	WinCC Professional 4096 PowerTags V11	6AV2103-0HA01-0AA5
	WinCC Professional max. PowerTags V11	6AV2103-0XA01-0AA5

C.9 Documentation

Table C- 15 S7-1200 documentation

Printed documentation	Language	Order Number
S7-1200 Programmable Controller System Manual	German	6ES7 298-8FA30-8AH0
	English	6ES7 298-8FA30-8BH0
	French	6ES7 298-8FA30-8CH0
	Spanish	6ES7 298-8FA30-8DH0
	Italian	6ES7 298-8FA30-8EH0
	Chinese	6ES7 298-8FA30-8KH0
S7-1200 Easy Book	German	6ES7 298-8FA30-8AQ0
	English	6ES7 298-8FA30-8BQ0
	French	6ES7 298-8FA30-8CQ0
	Spanish	6ES7 298-8FA30-8DQ0
	Italian	6ES7 298-8FA30-8EQ0
	Chinese	6ES7 298-8FA30-8KQ0

Index

A

ABS (absolute value), 190
AC
  grounding, 58
  inductive loads, 59
  isolation guidelines, 57
  wiring guidelines, 56, 58
Access protection
  CPU, 153
Accessing
  data logs from PC, 486
  user-defined Web pages, 510
ACOS (arc cosine or inverse cosine), 192
Active/passive communication
  configuring the partners, 118, 465
  connection IDs, 396
  parameters, 120
Active/Passive connection, 394
Ad hoc mode
  ISO on TCP, 400
  TCP, 400
ADD (add), 187
Add new device
  CPU, 110
  detect existing hardware, 111
  unspecific CPU, 111
Adding inputs or outputs to LAD or FBD instructions, 31
Addressing
  Boolean or bit values, 81
  individual inputs (I) or outputs (Q), 81
  memory areas, 81
  process image, 81
Air flow, 39
Alarm
  peripheral access, 440
Alarm interrupt, 281
Aliases in user-defined Web pages, 500
Analog I/O
  configuration, 115
  conversion to engineering units, 30, 85, 208
  input representation (voltage), 669, 678, 687, 708, 733
  output representation (current), 708, 733
  output representation (voltage), 709, 734
  status indicators, 634
  step response times (CPU), 669, 677, 686
  step response times (SB), 732
  step response times (SM), 707
Analog signal (SM)
  SM 1232 AQ 4 x 14bit, 702
Analog signal board (SB)
  SB 1231 AI 1 x 12 bit, 728
  SB 1231 AI 1 x 16 bit RTD, 738
  SB 1231 AI 1 x 16 bit Thermocouple, 734
  SB 1232 AQ 1x12 bit, 731
Analog signal module (SM)
  SM 1231 AI 4 x RTD x 16 bit, 714
  SM 1231 AI 8 x 13 bit, 700
  SM 1231 AI 8 x 16 bit TC, 709
  SM 1231 AI 8 x RTD x 16 bit, 714
  SM 1231 AI4 x 16 bit TC, 709
  SM 1232 AQ 2 x 14bit, 702
  SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit, 704
AND, 226
Approvals
  ATEX approval, 658
  CE approval, 657
  C-Tick approval, 659
  cULus approval, 658
  FM approval, 658
  maritime approval, 659
Arrays
  accessing members with a variable, 198
AS-i
  add AS-i slave, 453
  add CM 1243-2 AS-i Master module, 453
  AS-i address, 454
  AS-i address properties, 454
  Distributed I/O instructions, 458
  network connection, 453
  RDREC, 260
  system assignment, 457
  system assignment of slave addresses, 457
  transferring analog values, 455
  transferring digital values, 455
  WRREC, 260
AS-I
  slave configuration with STEP 7, 455
  slave configuration without STEP 7, 457
AS-i address, 454
  configuring, 454
AS-i Master Module, 452

CM 1243-2 AS-i Module Features, 452
ASIN (arc sine or inverse sine), 192
Assigning enum types, user-defined Web pages, 501
ATEX approval, 658
ATH (ASCII to hexadecimal), 251
ATTACH, 273
AWP commands, 494
combining definitions, 504
defining an enum type, 501
generating fragments, 503
importing fragments, 504
reading special variables, 497
referencing an enum type, 501
using an alias, 500
writing special variables, 499
writing variables, 496
AWP_Enum_Def, 501
AWP_Enum_Ref, 501
AWP_Import_Fragment, 504
AWP_In_Variable, 496, 499
AWP_Out_Variable, 497
AWP_Start_Fragment, 503

B

Basic panels (HMI), 25
Baud rate, 550
Binding to a CPU or memory card, 156
Bit logic, 163
Block
consistency check, 161
Block move (MOVE_BLK), 194
Blocks
block calls, 61
calling an FB or FC with SCL, 147
copying blocks from an online CPU, 157
counters (quantity and memory requirements), 19, 665, 673, 682
data block (DB), 61
download, 157
events, 68
function (FC), 61, 140
function block (FB), 61, 140
initial value of an FB, 140
instance data block (DB), 140
interrupts, 19, 68, 665, 673, 682
linear and structured programs, 136
monitoring, 19, 665, 673, 682
nesting depth, 19, 61, 665, 673, 682
number of code blocks, 19, 665, 673, 682
number of OBs, 19, 68, 665, 673, 682
organization blocks (OBs), 19, 61, 66, 68, 665, 673, 682
password protection, 154
single instance or multi-instance DB, 140
size of the user program, 19, 61, 665, 673, 682
start-up OBs, 68
timers (quantity and memory requirements), 19, 665, 673, 682
types of code blocks, 61
valid FC, FB, and DB numbers, 61
Boolean or bit values, 81
Break, 553, 554
BUFFER parameter, SEND_PTP, 543
Bus connector, 22

C

CALCULATE, 30, 186
scaling analogs, 30
Calendar, 233
Call structure, 161
Calling code blocks within the user program, 138
CAN_DINT (cancel time delay interrupt), 279
Capturing the status of a code block, 34
Capturing values from an online DB, 644
CB 1241 RS485 specifications, 753
CE approval, 657
CEIL (ceiling), 205
Certificate Import Wizard, 490
Changing settings for STEP 7, 33
Char (character data type), 90
Character position
message length, 559
Character sequence
message end, 558
message start, 554
Clearance
airflow and cooling, 39
Clock
memory byte, 79
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
time-of-day clock, 77
WR_SYS_T (write system time), 235
CM 1241
RS232 specifications, 756
RS422/RS485 specifications, 756
RS485 specifications, 754
CM 1243-2 AS-i
AS-i Module Features, 452
Code block
binding to a CPU or memory card, 156

block calls, 61
calling code blocks within the user program, 138
capturing the status of a code block, 34
copy protection, 156
counters (quantity and memory requirements), 19, 665, 673, 682
DB (data block), 61, 141
FB (function block), 61, 140
FC (function), 61, 140
initial value of an FB, 140
instance data block (DB), 140
interrupts, 19, 665, 673, 682
know-how protection, 154
linear and structured programs, 136
monitoring, 19, 665, 673, 682
nesting depth, 19, 665, 673, 682
number of code blocks, 19, 665, 673, 682
number of OBs, 19, 665, 673, 682
organization block (OB), 138
organization blocks (OBs), 19, 665, 673, 682
restoring the status of a code block, 34
size of the user program, 19, 665, 673, 682
timers (quantity and memory requirements), 19, 665, 673, 682
types of code blocks, 61
valid FC, FB, and DB numbers, 61
Code blocks, 138
Cold junction compensation
Thermocouple, 712, 736
Columns and headers in task cards, 32
Communication
active/passive, 118, 120, 465
AS-i address, 454
communication load, 74
configuration, 118, 120, 465
connection IDs, 396
cycle time, 74
flow control, 550
hardware connection, 429
IP address, 127
MAC address, 127
network, 429
network connection, 117
number of connections (PROFINET), 394
parameters, 120, 483
polling architecture, 561
PROFIBUS address, 449
PROFINET and PROFIBUS, 393
send and receive parameters, 552
statistics, 483
TCON_Param, 120
time synchronization property (PROFINET), 132
Communication board (CB)
add modules, 112
CB 1241 RS485, 753
comparison chart, 20
configuration of parameters, 115
device configuration, 109
installation, 46
LED indicators, 529, 633
overview, 22
programming, 560
removal, 46
RS485, 529
Communication interfaces
add modules, 112
CB 1241 RS485, 753
CM 1241 RS232, 756
CM 1241 RS485, 754
comparison chart of the modules, 20
configuration, 549
device configuration, 109
LED indicators, 633
programming, 560
RS232 and RS485, 529
Communication module
CM 1241 RS422/RS485 specifications, 756
Communication module (CM)
add CM 1243-2 AS-i Master module, 453
add CM 1243-5 (DP master) module, 448
add modules, 112
CM 1241 RS232 specifications, 756
CM 1241 RS485, 754
comparison chart, 20
configuration for PtP example program, 563
configuration of parameters, 115
data reception, 544
device configuration, 109
installation, 48
LED indicators, 529, 633
overview, 22
power requirements, 761
programming, 560
removal, 48
RS232 and RS485, 529
Communication processor (CP)
add modules, 112
comparison chart, 20
configuration of parameters, 115
device configuration, 109
overview, 22
Communication standard Web page, 483
Compare, 183
Comparing and synchronizing online/offline CPUs, 641

Comparing code blocks, 641
Comparison chart
  CPU models, 18
  HMI devices, 25
  modules, 20
Computer requirements, 27
CONCAT (concatenate), 254
Configuration
  add modules, 112
  AS-i, 454
  AS-i address, 454
  AS-i port, 454
  communication interfaces, 549
  communication load, 74
  CPU parameters, 114
  cycle time, 73
  discover, 111
  download, 157
  Ethernet port, 127
  HSC (high-speed counter), 323
  IP address, 127
  MAC address, 127
  modules, 115
  network connection, 117
  PLC to PLC communication, 433
  ports, 549
  PROFIBUS, 449
  PROFIBUS address, 449
  PROFIBUS port, 449
  PROFINET, 127
  receive message, 553
  startup parameters, 102
  time synchronization property (PROFINET), 132
  user-defined Web pages, 507
  user-defined Web pages (multiple languages), 525
Configuration of transmitted message, 552
Connection contacts
  Maximum current carrying capacity, 751
Connections
  configuration, 120
  connection IDs, 396
  Ethernet protocols, 464
  number of connections (PROFINET), 394
  partners, 118, 465
  S7 connection, 464
  types of communication, 393
  types, multi-node connections, 464
Connector
  installation and removal, 50
Consistency check, 161
Constraints
  user-defined Web pages, 510
  Web server, 488
Contact information, 3
Control DB for user-defined Web pages
  global commands, 526
  parameter to WWW instruction, 508
  request commands and states, 526
CONV (convert), 201
Conversion (SCL instructions), 202
Cookie restrictions, standard Web pages, 490
Cookie, siemens_automation_language, 523
Cooling, 39
Copy protection
  binding to a CPU or memory card, 156
Copying blocks from an online CPU, 157
COS (cosine), 192
Counters
  HSC (high-speed counter), 315
  HSC configuration, 323
  HSC operation, 317
  quantity, 19, 665, 673, 682
  size, 19, 665, 673, 682
CPU
  1211C specifications, 663
  1211C wiring diagrams, 671
  1212C specifications, 671
  1212C wiring diagrams, 680
  1214C specifications, 680
  1214C wiring diagrams, 688
  access protection, 153
  add modules, 112
  add new device, 110
  analog input representation (voltage), 669, 678, 687, 708, 733
  AS-i, 454
  AS-i address, 454
  AS-i port, 454
  assigning an IP address to an online CPU, 125
  capturing the status of a code block, 34
  capturing values of a DB, 644
  communication board (CB), 22
  communication load, 74
  comparing and synchronizing blocks, 641
  comparison chart, 18
  configuring communication to HMI, 431
  configuring multiple, 433
  configuring parameters, 114
  configuring pulse channels, 290
  configuring the modules, 115
  copying blocks from an online CPU, 157
  creating a program card, 105
  creating a transfer card, 102
  cycle time configuration, 74

cycle time monitoring, 73
device configuration, 109
displaying the MAC and IP addresses, 130
download, 157
download to device, 130
empty transfer card, 107
enable outputs in STOP mode, 647
Ethernet port, 127
expansion cable, 51
force, 647, 648
going online, 635
grounding, 58
HMI devices, 25
HSC configuration, 323
inductive loads, 59
inserting the memory card, 100
installation, 43, 44
IP address, 127
isolation guidelines, 57
know-how protection, 154
lamp loads, 59
LED indicators, 633
lost password, 107
MAC address, 127, 130
memory card, 99, 758
monitoring, 642
network connection, 117
number of communication connections, 394
online, 638, 642
operating modes, 63
operating panel (online CPU), 639
operator panel, 34
overview, 17
password protection, 153
power budget, 40
power requirements, 761
processing the OBs, 138
PROFIBUS, 449
PROFIBUS address, 449
PROFIBUS port, 449
PROFINET, 127
PROFINET IO, 437
program card, 99, 105
program execution, 61
pulse outputs, 289
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
recover from a lost password, 107
reset to factory settings, 638
resetting the start values of a DB, 644
restoring the status of a code block, 34
RUN and STOP mode, 639
run time meter, 237
RUN/STOP buttons, 34
Security levels, 153
signal board (SB), 22
startup parameters, 102
startup processing, 65
step response times, 669, 677, 686
terminal block connector, 50
thermal zone, 39, 42
time synchronization property, 132
transfer card, 99, 102
types of communication, 393
unspecific CPU, 111
watch table, 645
wiring guidelines, 56, 58
WR_SYS_T (write system time), 235
CPU communication, 395
CPU Identification standard Web page, 480
CPU properties, user-defined Web pages, 507
CPU properties, user-defined Web pages (multiple languages), 525
Creating a network connection, 117
Creating user-defined Web page DBs, 508
Creating user-defined Web pages, 493
Cross-references, 160
Introduction, 160
Uses, 160
CTD (count down), 178
C-Tick approval, 659
CTRL_PWM, 287
CTS, 550
CTU (count up), 178
CTUD (count up and down), 178
cULus approval, 658
Customer support, 3
Cycle time
configuration, 74
monitoring, 640
overview, 73
Cyclic-interrupt OB, 67

D

Data block
capturing values, 644
CONF_DATA, 423
global data block, 80, 141
instance data block, 80
organization block (OB), 138
overview, 61, 141
resetting the start values, 644
single FB with multiple instance DBs, 141

valid DB numbers, 61
data block control, 310
Data blocks for user-defined Web pages
importing fragments, 504
Data handling block (DHB), 141
Data log
Data log overview, 291
data record structure, 291
DataLogClose (close Data log), 297
DataLogCreate (create Data log), 293
DataLogNewFile (create Data log based on existing Data log), 300
DataLogOpen (open Data log), 296
DataLogWite (write Data log), 298
example program, 305
limits to Data log size, 303
viewing Data logs, 302
Data Logs standard Web page, 486
Data transmission, initiating, 541
Data types, 86
Any (pointer), 95
arrays, 91
Bool, Byte, Word, and DWord, 87
Char (character) and string, 90
PLC data type editor, 93
Pointer (pointer), 94
pointer data type overview, 93
Real, LReal (floating-point real), 88
Struc, 92
Time, Date, TOD (time of day), DTL (date and time long), 89
USInt, SInt, UInt, Int, UDInt, Dint (integer), 88
Variant (pointer), 96
Date
Date data type, 89
DTL (date and time long data type), 90
SET_TIMEZONE (set time zone), 238
T_ADD (add time), 234
T_COMBINE (combine times), 235
T_CONV (convert time), 233
T_DIFF (time difference), 234
T_SUB (subtract time), 234
DB (data block), 61, 141
capturing values, 644
resetting the start values, 644
valid DB numbers, 61
DC
grounding, 58
inductive loads, 59
isolation guidelines, 57
wiring guidelines, 56, 58
Debugging
downloading in RUN mode, 650, 654
DEC (decrement), 189
DECO (decode), 227
Defining enum types, user-defined Web pages, 501
DELETE (delete substring), 256
Designing a PLC system, 135, 136
DETACH, 273
Device
PROFINET IO, 437
Device configuration, 109, 430
add modules, 112
add new device, 110
AS-i, 454
AS-i port, 454
configuring the CPU, 114
configuring the modules, 115
discover, 111
download, 157
Ethernet port, 127
network connection, 117
PROFIBUS, 449
PROFIBUS port, 449
PROFINET, 127
time synchronization property (PROFINET), 132
unplugged modules, 38
Device names
PROFINET IO, 438
DeviceStates, 283
Diagnostic error interrupt OB, 67
Diagnostic standard Web page, 480
Diagnostics
cycle time, 640
DeviceStates, 283
diagnostics buffer, 640
GET_DIAG, 285
interrupt OB, 440
LED indicators, 633
LED instruction, 282
memory usage, 640
ModuleStates, 284
status, 440
status indicator, 79
watch table, 645
Diagnostics buffer, 77, 640
Digital I/O
configuration, 115
status indicators, 634
Digital signal board (SB) specifications
SB 1221 DI 4, 200 kHz, 720
SB 1222 DQ 4, 200 kHz, 722
SB 1223 DI 2 / DQ 2, 726
SB 1223 DI 2 / DQ 2, 200 kHz, 724

Digital signal module (SM)
SM 1221, 689
SM 1222, 691, 693
SM 1223, 696
DIN rail, 43
Directories, languages for user-defined Web pages, 522
DIS_AIRT (disable alarm interrupt), 281
Discover to upload an online CPU, 111
Displaying the MAC and IP addresses, 130
DIV (divide), 187
Documentation, 4
Downloading
displaying the MAC and IP addresses, 130
project, 157
Siemens security certificate to PC, 478, 490
user program, 157
user-defined Web page DBs, 509
DPNRM_DG, 271
DPRD_DAT, 269
DPWR_DAT, 269
Drag and drop between editors, 33
DTL data type
system clock instructions, 235

E

Electromagnetic compatibility (EMC), 660
EN and ENO (power flow), 152
EN_AIRT (enable alarm interrupt), 281
ENCO (encode), 227
End conditions, 556
End message character, 558
Enum types in user-defined Web pages, 501
Environmental
industrial environments, 659
operating conditions, 660
transport and storage conditions, 660
Error codes
common errors for extended instructions, 313
RALRM, 265
RDREC, 265
WRREC, 265
Errors
diagnostic errors, 71
time errors, 70
Ethernet
ad hoc mode, 400
connection IDs, 396
DPNRM_DG, 271
DPRD_DAT, 269
DPWR_DAT, 269
GET, 460
IP address, 127
MAC address, 127
network connection, 117
number of communication connections, 394
overview, 399
PUT, 460
RALRM, 263
RDREC, 260
T_CONFIG, 421
TCON, 408
TDISCON, 408
TRCV, 408
TRCV_C, 401
TSEND, 408
TSEND_C, 401
TURCV, 416
TUSEND, 416
types of communication, 393
WRREC, 260
Ethernet protocols, 399
multi-node connections, 464
Event execution, 68
Example
Modbus slave, 617
PtP communication, 562
PtP communication, configuration, 563
PtP communication, running, 567
PtP communication, STEP 7 programming, 565
PtP communication, terminal emulator, 566
user-defined Web pages, 511
EXP (natural exponential), 192
Expandable instructions, 32
Expanding the capabilities of the S7-1200, 20
Expansion cable
installation, 51
removal, 51
EXPT (general exponential), 192

F

Factory settings reset, 638
FAQs, 4
Favorites toolbar, 29
FB (function block)
overview, 61
valid FB numbers, 61
FBD (function block diagram), 145
FC (function), 61, 140
valid FC numbers, 61
FieldRead, 197
FieldWrite, 197

FILL_BLK, 199
FIND (find substring), 259
First scan indicator, 79
Floating-point math, 192
FLOOR, 205
Flow control, 550
  configuration, 550
FM approval, 658
Folders, languages for user-defined Web pages, 522
Force, 647, 648
  I memory, 647, 648
  inputs and outputs, 648
  memory card does not contain force values, 99
  peripheral inputs, 647, 648
  scan cycle, 648
  watch table, 645
Force table
  addressing peripheral inputs, 647
  force, 647
  force operation, 648
FRAC (fraction), 192
Fragment DBs (user-defined Web pages)
  generating, 508
Fragments (user-defined Web pages)
  creating from AWP command, 503
  importing with AWP command, 504
Freeport protocol, 531
Frequency, clock bits, 79
Function (FC)
  calling code blocks within the user program, 138
  capturing the status of a code block, 34
  know-how protection, 154
  linear and structured programs, 136
  overview, 61, 140
  restoring the status of a code block, 34
  valid FC numbers, 61
Function block (FB)
  calling code blocks within the user program, 138
  capturing the status of a code block, 34
  initial value, 140
  instance data block, 140
  know-how protection, 154
  linear and structured programs, 136
  output parameters, 140
  overview, 61, 140
  restoring the status of a code block, 34
  single FB with multiple instance DBs, 141
  valid FB numbers, 61

G

General technical specifications, 657
Generating user-defined Web page DBs, 508
GET, 460
  configuring the connection, 119
Get LED status, 282
GET_DIAG, 285
GetError, 223
GetErrorID, 224
Global data block, 80, 141
Global library
  USS protocol overview, 567
Guidelines
  CPU installation, 44
  grounding, 58
  inductive loads, 59
  installation, 39
  installation procedures, 43
  isolation, 57
  lamp loads, 59
  wiring guidelines, 56, 58

H

Hardware configuration, 109
  add modules, 112
  add new device, 110
  AS-i, 454
  AS-i port, 454
  configuring the CPU, 114
  configuring the modules, 115
  discover, 111
  download, 157
  Ethernet port, 127
  network connection, 117
  PROFIBUS, 449
  PROFIBUS port, 449
  PROFINET, 127
Hardware flow control, 550
Hardware-interrupt OB, 67
High-speed counter
  configuration, 323
  HSC, 315
  operation, 317
High-speed counter (HSC)
  cannot be forced, 649
HMI devices
  configuring PROFINET communication, 431
  network connection, 117
  overview, 25
Hotline, 3
HSC (high-speed counter)
  configuration, 323
  operation, 315, 317

HTA (hexadecimal to ASCII), 252
HTML listing, user-defined Web page example, 516
HTML pages, user-defined, 492
  accessing S7-1200 data, 494
  developing, 493
  language locations, 525
  page locations, 507
  refreshing, 494

I

I memory
  force, 647
  force operation, 648
  force table, 647
  HSC (high-speed counter), 317
  monitor, 642
  monitor LAD, 643
  peripheral input addresses (force table), 647
  watch table, 642
I/O
  access errors, PROFINET, 441
  addressing, 85
  analog input representation (voltage), 669, 678, 687, 708, 733
  analog output representation (current), 708, 733
  analog output representation (voltage), 709, 734
  analog status indicators, 634
  digital status indicators, 634
  force, 647
  force operation, 648
  inductive loads, 59
  monitoring status in LAD, 643
  monitoring with a watch table, 645
  step response times (CPU), 669, 677, 686
  step response times (SB), 732
  step response times of the signal module (SM), 707
Identification standard Web page, 480
Idle line, 553, 554
Importing Siemens security certificate, 490
IN_RANGE (within a range), 184
INC (increment), 189
Indexing arrays with variables, 198
Inductive loads, 59
Information resources, 4
Initial values
  capturing and resetting the start values of a DB, 644
Input simulators, 759
Inputs and outputs
  monitoring, 642
INSERT (insert substring), 257
Inserting a device
  unspecific CPU, 111
Inserting instructions
  drag and drop, 29
  drag and drop between editors, 33
  favorites, 29
Inserting the memory card into CPU, 100
Installation
  air flow, 39
  clearance, 39
  communication board (CB), 46
  communication module (CM), 48
  cooling, 39
  CPU, 44
  expansion cable, 51
  grounding, 58
  guidelines, 39
  inductive loads, 59
  isolation guidelines, 57
  lamp loads, 59
  mounting dimensions, 42
  overview, 39, 43
  power budget, 40
  signal board (SB), 46
  signal module (SM), 22, 47
  terminal block connector, 50
  thermal zone, 39, 42
  TS Adapter and TS module, 53
  TS Adapter on a DIN rail, 54
  TS Adapter on a wall, 56
  TS Adapter SIM card, 53
  wiring guidelines, 56, 58
Installation requirements, 27
Instance data block, 80
Instructions
  ABS (absolute value), 190
  ACOS (arc cosine or inverse cosine), 192
  ADD (add), 187
  adding inputs or outputs to LAD or FBD instructions, 31
  AND, 226
  AS-i Distributed I/O, 458
  ASIN (arc sine or inverse sine), 192
  ATAN (arc tangent or inverse tangent), 192
  ATH (ASCII to hexadecimal), 251
  ATTACH, 273
  bit logic, 163
  block move (MOVE_BLK), 194
  CALCULATE, 30, 186
  calendar, 233
  CAN_DINT (cancel time delay interrupt), 279
  CASE (SCL), 211
  CEIL (ceiling), 205

clock, 235
columns and headers, 32, 599
common parameters, 427
compare, 183
CONCAT (concatenate), 254
CONTINUE (SCL), 214
CONV (convert), 201
COS (cosine), 192
counters, 178
CTD (count down), 178
CTRL_PWM), 287
CTU (count up), 178
CTUD (count up and down), 178
data block control, 310
DataLogClose (close Data log), 297
DataLogCreate (create Data log), 293
DataLogNewFile (create Data log based on existing Data log), 300
DataLogOpen (open Data log), 296
DataLogWite (write Data log), 298
date, 233
DEC (decrement), 189
DECO (decode), 227
DELETE (delete substring), 256
DETACH, 273
DeviceStates, 283
DIS_AIRT (disable alarm interrupt), 281
DIV (divide), 187
DPNRM_DG, 271
DPRD_DAT, 269
DPWR_DAT, 269
drag and drop, 29
drag and drop between editors, 33
EN_AIRT (enable alarm interrupt), 281
ENCO (encode), 227
EXIT (SCL), 215
EXP (natural exponential), 192
expandable instructions, 32
EXPT (general exponential), 192
favorites, 29
FieldRead, 197
FieldWrite, 197
FILL_BLK, 199
FIND (find substring), 259
floating-point math, 192
FLOOR, 205
FOR (SCL), 212
force, 647
force operation, 648
FRAC (fraction), 192
GET, 460
GET_DIAG, 285
GetError, 223
GetErrorID, 224
GO TO (SCL), 216
HSC (high-speed counter), 315, 317
HTA (Hex to ASCII), 252
IF-THEN (SCL), 210
IN_RANGE (within a range), 184
INC (increment), 189
INSERT (insert substring), 257
inserting, 29
INV (invert), 227
JMP, 217
JMP_LIST, 217
Label, 217
LED status, 282
LEFT (left substring), 255
LEN (length), 253
limit, 192
LN (natural logarithm), 192
MAX (maximum), 191
MB_CLIENT, 584
MC_ChangeDynamic, 365
MC_CommandTable, 362
MC_Halt, 352
MC_Home, 350
MC_MoveAbsolute, 354
MC_MoveJog, 360
MC_MoveRelative, 356
MC_MoveVelocity, 358
MC_Power, 346
MC_Reset, 349
MID (middle substring), 255
MIN (minimum), 191
MOD (modulo), 188
ModuleStates, 284
monitor, 642, 643
MOVE, 194
MUL (multiply), 187
MUX (multiplex), 229
N_TRIG, 169
NEG (negation), 189
negative edge, 168
NORM_X (normalize), 206
NOT OK, 185
OK, 185
OR, 226
OUT_RANGE (outside of a range), 184
P_TRIG, 169
PID_Compact, 328
PORT_CFG (port configuration), 534
positive edge, 168
PROFIBUS distributed I/O, 451

PROFINET Distributed I/O, 441
program control (SCL), 209
PUT, 460
QRY_CINT (query cyclic interrupt), 278
RALRM, 263
RCV_CFG (receive configuration), 537
RCV_PtP (receive Point-to-Point), 544
RCV_RST (receiver reset), 546
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
RDREC, 260
RE_TRIGR, 73, 221
REPEAT (SCL), 214
REPLACE (replace substring), 258
reset, 166
RETURN (SCL), 216
return value (RET), 220
RIGHT (right substring), 255
ROL and ROR (rotate left and rotate right), 232
ROUND, 204
RT (reset timer), 170
run time meter, 237
S_CONV (value to string conversions), 241
S_MOV (string move), 240
SCALE_X (scale), 206
scaling analog values, 30
SCL conversion instructions, 202
SEL (select), 228
SEND_CFG (send configuration), 535
SEND_PTP (send Point-to-Point data), 541
Set, 166
SET_CINT (set cyclic interrupt), 276
SET_TIMEZONE, 238
SGN_GET (get RS232 signals), 547
SGN_SET (set RS232 signals), 548
SHL and SHR (shift left and shift right), 231
SIN (sine), 192
SQR (square), 192
SQRT (square root), 192
SRT_DINT (start time delay interrupt), 279
status, 642, 643
STP (stop PLC scan cycle), 222
STRG_VAL (string to value), 241
SUB (subtract), 187
SWAP, 200
SWITCH, 218
T_ADD (add time), 234
T_COMBINE (combine times), 235
T_CONFIG, 421
T_CONV (convert time), 233
T_DIFF (time difference), 234
T_SUB (subtract time), 234
TAN (tangent), 192
TCON, 408
TDISCON, 408
time, 233
timer, 170
timer operations, 174
TOF (off-delay timer), 170
TON (on-delay timer), 170
TONR (on-delay retentive timer), 170
TP (pulse timer), 170
TRCV, 408
TRCV_C, 401, 436
TRUNC (truncate), 204
TSEND, 408
TSEND_C, 401, 435
TURCV, 416
TUSEND, 416
UFILL_BLK (uninterruptible fill), 199
uninterruptible move (UMOVE_BLK), 194
USS status codes, 576
USS_DRV, 570
USS_PORT, 573
USS_RPM, 574
USS_WPM, 575
VAL_STRG (value to string), 241
versions of instructions, 32, 599
WHILE (SCL), 213
WR_SYS_T (write system time), 235
WRREC, 260
WWW (enable user-defined Web pages), 508
XOR (exclusive OR), 226
Inter-character gap, 558
Interrupts
ATTACH and DETACH, 273
CAN_DINT (cancel time delay interrupt), 279
interrupt latency, 68
overview, 66
SRT_DINT (start time delay interrupt), 279
Intro standard Web page, 478
Invert (INV), 227
IP address, 127, 128
assigning, 123, 129
assigning online, 125
configuring, 127
configuring the online CPU, 638
device configuration, 114
MAC address, 127
IP router, 127
ISO on TCP
ad hoc mode, 400
ISO on TCP protocol, 399
Isolation guidelines, 57

ISO-on-TCP
connection configuration, 118
connection IDs, 396
parameters, 120

J

JavaScript restrictions, standard Web pages, 489
JMP, 217
JMP_LIST, 217
JMPN, 217

K

Know-how protection
password protection, 154

L

Label, 217
LAD (ladder logic)
monitor, 642, 643
overview, 144
program editor, 643
status, 642, 643, 647
Lamp loads, 59
Languages, user-defined Web pages, 522
Latency, 68
LED (Get LED status), 282
LED indicators
communication interface, 529, 633
CPU status, 633
LED instruction, 282
LEFT (left substring), 255
LEN (length), 253
Length
message, 559
Length m, 559
Length n, 559
LENGTH parameter, SEND_PTP, 543
Limit, 192
Linear programming, 136
LN (natural logarithm), 192
Load memory, 18
CPU 1211C, 663
CPU 1212C, 671
CPU 1214C, 680
memory card, 99
program card, 99
transfer card, 99
Load memory, user-defined Web pages, 510
Local time
RD_LOC_T (read local time), 235
Local/Partner connection, 394
Logging in/out
standard Web pages, 477
Lost password, 107

M

MAC address, 127, 130
Manual fragment DB control, 526
Manuals, 4
Maritime approval, 659
Master polling architecture, 561
Math, 30, 186, 187
MAX (maximum), 191
Maximum message length, 558
MB_CLIENT, 584
MB_COMM_LOAD, 600
MB_MASTER, 603
MB_SERVER, 590
MB_SLAVE, 608
MC_ChangeDynamic, 365
MC_CommandTable, 362
MC_Halt, 352
MC_Home, 350
MC_MoveAbsolute, 354
MC_MoveJog, 360
MC_MoveRelative, 356
MC_MoveVelocity, 358
MC_Power, 346
MC_Reset, 349
Memory
clock memory, 78
I (process image input), 82
L (local memory), 80
load memory, 75
M (bit memory), 83
memory card does not contain force values, 99
monitoring memory usage, 640
peripheral input addresses (force table), 647
Q (process image output), 83
retentive memory, 75
system memory, 78
Temp memory, 84
work memory, 75
Memory areas
addressing Boolean or bit values, 81
immediate access, 81
process image, 81
Memory card
configure the startup parameters, 102

empty transfer card for a lost password, 107
inserting into CPU, 100
lost password, 107
operation, 99
order number, 758
overview, 99
program card, 105
specifications, 758
transfer card, 102
Memory locations, 80, 82
Message configuration
instructions, 561
receive, 553
transmit, 552
Message end, 556
Message length, 558
Message start, 553
MID (middle substring), 255
MIN (minimum), 191
Miscellaneous PtP parameter errors, 533
MOD (modulo), 188
MODBUS
MB_CLIENT, 584
MB_COMM_LOAD, 600
MB_MASTER, 603
MB_SERVER, 590
MB_SLAVE, 608
Modbus slave example, 617
versions, 32, 599
Modifying
memory card does not contain force values, 99
program editor status, 643
watch table, 645
Modifying variables from PC, 484
Module information standard Web page, 481
Modules
CB 1241 RS485, 753
CM 1241 RS232, 756
CM 1241 RS485, 754
communication board (CB), 22
communication module (CM), 22
communication processor (CP), 22
comparison chart, 20
configuring parameters, 115
CPU 1211C specifications, 663
CPU 1212C specifications, 671
CPU 1214C specifications, 680
SB 1221 DI 4, 200 kHz, 720
SB 1222 DQ 4, 200 kHz, 200 kHz, 722
SB 1223 DI 2 / DQ 2, 726
SB 1223 DI 2 / DQ 2, 200 kHz, 724
SB 1231 AI 1 x 12 bit, 728
SB 1231 AI 1 x 16 bit RTD, 738
SB 1231 AI 1 x 16 bit Thermocouple signal board, 734
SB 1232 AQ 1x12 bit, 731
signal board (SB), 22
signal module (SM), 22
SM 1221, 689
SM 1222, 691, 693
SM 1222 DQ8 RLY Changeover, 691
SM 1223, 696
SM 1231 AI 4 x 16 bit TC, 709
SM 1231 AI 4 x RTD x 16 bit, 714
SM 1231 AI 8 x 13 bit, 700
SM 1231 AI 8 x 16 bit TC signal module, 709
SM 1231 AI 8 x RTD x 16 bit, 714
SM 1232 AQ 2 x 14bit, 702
SM 1232 AQ 4 x 14bit, 702
SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit, 704
thermal zone, 39, 42
ModuleStates, 284
Monitor
capturing values of a DB, 644
resetting the start values of a DB, 644
Monitoring
cycle time, 640
force operation, 648
force table, 647
LAD status, 642, 643
LED instruction, 282
memory card does not contain force values, 99
memory usage, 640
watch table, 642, 645
Monitoring the program, 159
Monitoring variables from PC, 484
Motion control
configuring the axis, 343
hardware and software limit switches, 368
homing (sequence for active homing), 374
homing configuration parameters, 372
homing the axis, 371
MC_ChangeDynamic, 365
MC_CommandTable, 362
MC_Halt, 352
MC_Home, 350
MC_MoveAbsolute, 354
MC_MoveJog, 360
MC_MoveRelative, 356
MC_MoveVelocity, 358
MC_Power, 346
MC_Reset, 349
overview, 340
Mounting

airflow, 39
clearance, 39
communication board (CB), 46
communication module (CM), 48
cooling, 39
CPU, 44
dimensions, 42
expansion cable, 51
grounding, 58
guidelines, 39
inductive loads, 59
isolation, 57
lamp loads, 59
overview, 43
signal board (SB), 46
signal module (SM), 47
terminal block connector, 50
thermal zone, 39, 42
wiring guidelines, 56, 58
MOVE, 194
MRES
operator panel, 34
MUL (multiply), 187
Multi-node connections
connection types, 464
Ethernet protocols, 464
Multiple AWP variable definitions, 504
MUX (multiplex), 229
My Documentation Manager, 4

N

N_TRIG, 169
NEG (negation), 189
Negative edge, 168
Nesting depth, 61
Network communication, 429
Network connection
configuration, 117
multiple CPUs, 432, 434, 438, 449, 453
Network time protocol (NTP), 131
No restart, 63
NORM_X (normalize), 206
Normalizing analogs, 208
NOT OK instruction, 185
Numbers
binary, 87
integer, 88
real, 88

O

Off-delay (TOF), 170
operation, 174
OK instruction, 185
On-delay delay (TON), 170
operation, 174
On-delay retentive (TONR), 170
operation, 174
Online
assigning an IP address, 125
capturing values of a DB, 644
comparing and synchronizing, 641
cycle time, 640
diagnostics buffer, 640
force, 647
force operation, 648
going online, 635
IP address, 638
memory usage, 640
monitor, 642
operating panel, 639
operator panel, 34
resetting the start values of a DB, 644
RUN/STOP buttons, 34
status, 642, 643
time of day, 638
tools, 642
watch table, 642, 643, 645
Online and diagnostic tools
downloading in RUN mode, 650
Online device names
PROFINET IO, 636
OPC, 624
Open User Communication instructions return values, 428
Operating mode, 34
changing STOP/RUN, 639
operating modes of the CPU, 63
Operator panel, 34
operating modes of the CPU, 63
Operator panels, 25
OR, 226
Order numbers
Communication interfaces (CM, CB and CP), 766, 767
connector blocks, 768
Connectors and terminal connections, 767
CPUs, 765
CSM 1277 Ethernet switch, 767
documentation, 769
Expansion cables, 768
HMI basic panels, 768

memory cards, 768
PM 1207 power supply, 767
programming software, 769
Signal boards (SB), 765
Signal modules (SM), 766
simulators, 768
STEP 7, 769
visualization software, 769
WinCC, 769
Organization block
call, 66
calling code blocks within the user program, 138
capturing the status of a code block, 34
configuring operation, 140
creating, 139
function, 66
know-how protection, 154
linear and structured programming, 136
multiple cyclic, 139
overview, 61
priority classes, 66
processing, 138
restoring the status of a code block, 34
startup processing, 65
OUT_RANGE (outside of a range), 184
Output parameters, 140
configuring pulse channels, 290
pulse outputs, 289

P

P_TRIG, 169
Panels (HMI), 25
Parameter assignment, 140
Parameters configuration
LENGH and BUFFER for SEND_PTP, 543
receive, 437
transmit, 436
Parity, 550
Passive/active communication
configuring the partners, 118, 465
connection IDs, 396
parameters, 120
Password protection
access to the CPU, 153
binding to a CPU or memory card, 156
code block, 154
copy protection, 156
CPU, 153
empty transfer card, 107
lost password, 107
Peripheral access alarms, 440
PID
overview, 324
PID_3STEP, 331
PID_3Step algorithm, 325, 331
PID_Compact, 328
PID_Compact algorithm, 325, 328
PLC
add modules, 112
assigning an IP address to an online CPU, 125
communication load, 74
comparing and synchronizing, 641
copying blocks from an online CPU, 157
CPU 1211C, 663
CPU 1212C, 671
CPU 1214C, 680
cycle time, 74
cycle time, 74
device configuration, 109
download, 157
expansion cable, 51
force, 647
force operation, 648
HSC configuration, 323
installation, 43, 44
know-how protection, 154
memory card, 99, 758
monitoring, 642
operating modes, 63
overview of the CPU, 17
power budget, 40
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
run time meter, 237
startup processing, 65
system design, 135
tags, 80
terminal block connector, 50
time synchronization property, 132
using blocks, 136
watch table, 645
WR_SYS_T (write system time), 235
Podcasts, 4
Pointer
pointer overview, 93
Pointers
Any data type, 95
Pointer data type, 94
Variant data type, 96
Point-to-Point communication, 531
Point-to-Point programming, 560
Polling architecture, 561
Port configuration, 549

errors, 535
instructions, 561
PtP example program, 563
Port number, 399
Port numbers
restricted, 429
PORT_CFG (port configuration), 534
Portal view, 28
Positive edge, 168
Power budget, 40
example, 762
form for calculations, 763
overview, 761
Power requirements
calculating a power budget, 762
form for calculations, 763
power budget, 761
Priority
priority class, 66
priority in processing, 68
Priority class, 66
Process image
force, 647
force operation, 648
monitor, 642, 643
status, 642, 643, 647
PROFIBUS
add CM 1243-5 (DP master) module, 448
add DP slave, 448
CM 1242-5 (DP slave) module, 443
CM 1243-5 (DP master) module, 443
distributed I/O instructions, 451
DPNRM_DG, 271
DPRD_DAT, 269
DPWR_DAT, 269
GET, 460
master, 442
network connection, 117, 449
PROFIBUS address, 449
PROFIBUS address properties, 450
PUT, 460
RALRM, 263
RDREC, 260
S7 connection, 464
slave, 442
WRREC, 260
PROFIBUS address, 449, 450
configuring, 449
PROFINET
ad hoc mode, 400
configuring communication between CPU and HMI device, 431
configuring the IP address, 114
connection IDs, 396
CPU-to-CPU communication, 433
device naming and addressing, 133
diagnostics, 441
Distributed I/O instructions, 441
DPRD_DAT, 269
DPWR_DAT, 269
Ethernet address properties, 128
GET, 460
IP address, 127
IP address assignment, 133
MAC address, 127
network connection, 117, 432, 434, 438
number of communication connections, 394
overview, 399
PLC-to-PLC communication, 433
PUT, 460
RALRM, 263
RDREC, 260
S7 connection, 464
system start-up time, 132
T_CONFIG, 421
TCON, 408
TDISCON, 408
testing a network, 129
time synchronization, 114
time synchronization property, 132
TRCV, 408
TRCV_C, 401
TSEND, 408
TSEND_C, 401
TURCV, 416
TUSEND, 416
types of communication, 393
WRREC, 260
PROFINET IO
Adding a device, 437
Assigning a CPU, 438
Assigning device names, 438
Assigning device names online, 636
Device names, 438
Devices, 437
Online device names, 636
PROFINET RT, 399
Program
binding to a CPU or memory card, 156
calling code blocks within the user program, 138
capturing the status of a code block, 34
capturing values of a DB, 644
copying blocks from an online CPU, 157
download, 157

linear and structured programs, 136
memory card, 99
organization block (OB), 138
password protection, 154
priority class, 66
resetting the start values of a DB, 644
restoring the status of a code block, 34
Program card
configure the startup parameters, 102
creating, 105
inserting into CPU, 100
operation, 99
order number, 758
overview, 99
Program control (SCL), 209
CASE, 211
CONTINUE, 214
EXIT, 215
FOR, 212
GO TO, 216
IF-THEN, 210
REPEAT, 214
RETURN, 216
WHILE, 213
Program editor
capturing the status of a code block, 34
capturing values of a DB, 644
monitor, 643
resetting the start values of a DB, 644
restoring the status of a code block, 34
status, 643
Program execution, 61
Program information
In the call structure, 161
Program structure, 138
calling code blocks, 138
Programming
adding inputs or outputs to LAD or FBD instructions, 31
binding to a CPU or memory card, 156
block calls, 61
calling code blocks within the user program, 138
comparing and synchronizing code blocks, 641
counters, 178
data block (DB), 61
drag and drop between editors, 33
expandable instructions, 32
favorites, 29
FBD (function block diagram), 145
function (FC), 140
function block (FB), 61, 140
initial value of an FB, 140
inserting instructions, 29
instance data block (DB), 140
LAD (ladder), 144
linear program, 136
operating modes of the CPU, 63
organization block (OB), 138
PID overview, 324
PID_3STEP, 331
PID_3Step algorithm, 325, 331
PID_Compact, 328
PID_Compact algorithm, 325, 328
power flow (EN and ENO), 152
priority class, 66
PtP instructions, 560
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
run time meter, 237
SCL (Structured Control Language), 145, 146, 147
structured program, 136
types of code blocks, 61
unplugged modules, 38
unspecific CPU, 111
valid FC, FB, and DB numbers, 61
WR_SYS_T (write system time), 235
Programming user-defined Web page language switch, 523
Project
access protection, 153
binding to a CPU or memory card, 156
comparing and synchronizing, 641
download, 157
empty transfer card, 107
lost password, 107
program card, 105
protecting a code block, 154
restricting access to a CPU, 153
transfer card, 102
Project view, 28
Protection class, 661
Protection level
binding to a CPU or memory card, 156
code block, 154
CPU, 153
lost password, 107
Protocol
communication, 531
freeport, 531
ISO on TCP, 399
Modbus, 531
PROFINET RT, 399
TCP, 399
UDP, 399

USS, 531
PTO (pulse train output)
cannot be forced, 649
configuring pulse channels, 290
CTRL_PWM, 287
operation, 289
PtP communication, 531
configuring parameters, 552
configuring ports, 549
example program, 562
example program configuration, 563
example program, running, 567
example program, STEP 7 programming, 565
programming, 560
terminal emulator for example program, 566
PtP error classes, 533
PtP instruction return values, 532
Pulse delay (TP), 170
operation, 174
Pulse outputs, 289
PUT, 460
configuring the connection, 119
PWM (pulse width modulation)
cannot be forced, 649
configuring pulse channels, 290
CTRL_PWM, 287
operation, 289

Q

Q memory
configuring pulse channels, 290
pulse outputs, 289
QRY_CINT (query cyclic interrupt), 278
Queuing, 68
Quotation mark conventions, Web server, 505

R

RALRM, 263, 265
Rated voltages, 661
RCV_CFG (receive configuration), 537
RCV_PTP (receive Point-to-Point), 544
RCV_RST (receiver reset), 546
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
RDREC, 260, 265
RE_TRIGR, 221
READ_DBL, 310
Reading HTTP variables, 497
Receive configuration errors, 541
Receive message configuration, 553
PtP example program, 564
Receive parameters configuration, 437
Receive runtime return values, 544
Referencing enum types, user-defined Web pages, 501
Refreshing user-defined Web pages, 494
Relay electrical service life, 663
REPLACE (replace substring), 258
Replacing modules, 38
Requirements, installation, 27
Reset, 166
Reset timer (RT), 170
Reset to factory settings, 638
Resetting the start values of a DB, 644
Restoring the status of a code block, 34
Restricted TSAPs and port numbers, 429
Retentive memory, 18, 75
CPU 1211C, 663
CPU 1212C, 671
CPU 1214C, 680
Return value (RET), 220
Return values
Open User Communication instructions, 428
PtP instructions, 532
RIGHT (right substring), 255
ROL and ROR (rotate left and rotate right), 232
ROUND, 204
Router IP address, 128
RS232 and RS485 communication modules, 529
RT (reset timer), 170
RTS, 550
RTS always on, 551
RTS Off delay, 552
RTS On delay, 552
RTS switched, 550
RUN mode, 63, 66, 639
force operation, 648
operator panel, 34
toolbar buttons, 34
Run time meter, 237
RUN to STOP transition, 80
RUN/STOP buttons, 34

S

S_CONV (value to string conversions), 241
S_MOV (string move), 240
S7 communication
configuring the connection, 119
S7-1200
access protection, 153
add modules, 112

add new device, 110
airflow, 39
AS-i, 454
AS-i address, 454
AS-i port, 454
capturing the status of a code block, 34
capturing values of a DB, 644
clearance, 39
communication board (CB), 22
communication load, 74
communication module (CM), 22
communication processor (CP), 22
compare code blocks, 641
comparison chart of CPU models, 18
configuring the CPU parameters, 114
configuring the modules, 115
cooling, 39
CPU installation, 44
cycle time, 74
device configuration, 109
empty transfer card for a lost password, 107
Ethernet port, 127
expansion cable, 51
force, 647
force operation, 648
grounding, 58
HMI devices, 25
HSC configuration, 323
inductive loads, 59
installation, 43
installing a CB, 46
installing a CM, 48
installing an SB, 46
installing an SM, 47
IP address, 127
isolation guidelines, 57
know-how protection, 154
lamp loads, 59
lost password, 107
MAC address, 127
memory card, 758
modules, 20
monitoring, 642
mounting dimensions, 42
network connection, 117
operating modes, 63
operation, 645
operator panel, 34
overview of the CPU, 17
password protection, 153
power budget, 40
PROFIBUS, 449
PROFIBUS address, 449
PROFIBUS port, 449
PROFINET, 127
program card, 105
pulse outputs, 289
resetting the start values of a DB, 644
restoring the status of a code block, 34
RUN/STOP buttons, 34
signal board (SB), 22
signal module (SM), 22
startup parameters, 102
startup processing, 65
terminal block connector, 50
thermal zone, 39, 42
transfer card, 102
TS Adapter, 20
wiring guidelines, 56, 58
SB 1221
SB 1221 DI 4, 200 kHz wiring diagram, 721
SB 1222
SB 1222 DQ 4 x 24 VDC, 200 kHz wiring diagram, 723
SB 1223
SB 1223 DI 2 / DQ 2 wiring diagram, 728
SB 1223 DI 2 / DQ, 200 kHz wiring diagram, 726
SB 1231 AI 1 x 16 bit Thermocouple
Filter selection table, 737
SB 1232
SB 1232 AQ 1 x 12 bit wiring diagram, 731
SCALE_X (scale), 206
Scaling analogs, 30, 208
Scan cycle
force, 647
force operation, 648
Scan cycle time
overview, 73
SCL (Structured Control Language)
ABS (absolute value), 190
ACOS (arc cosine or inverse cosine), 192
addressing, 147
AND, 226
ASIN (arc sine or inverse sine), 192
ATAN (arc tangent or inverse tangent), 192
ATH (ASCII to hexadecimal), 251
ATTACH and DETACH, 273
bit logic, 163
calling an FB or FC, 147
calling blocks, 138
CAN_DINT (cancel time delay interrupt), 279
CASE, 211
CEIL (ceiling), 205
compare, 183

CONCAT (concatenate), 254
conditions, 147
CONTINUE, 214
control statements, 147, 209, 210, 211, 212, 213, 214, 215, 216
CONV (convert), 201
Conversion instructions, 202
COS (cosine), 192
counters, 178
CTD (count down), 178
CTU (count up), 178
CTUD (count up and down), 178
DataLogClose (close Data log), 297
DataLogCreate (create Data log), 293
DataLogNewFile (create Data log based on existing Data log), 300
DataLogOpen (open Data log), 296
DataLogWrite (write Data log), 298
DEC (decrement), 189
DECO (decode), 227
DELETE (delete substring), 256
DeviceStates, 283
DIS_AIRT (disable alarm interrupt), 281
EN and ENO (power flow), 152
EN_AIRT (enable alarm interrupt), 281
ENCO (encode), 227
EXIT, 215
EXP (natural exponential), 192
expressions, 147
EXPT (general exponential), 192
FILL_BLK, 199
FIND (find substring), 259
floating-point math, 192
FLOOR, 205
FOR, 212
FRAC (fraction), 192
GET_DIAG, 285
GO TO, 216
HTA (hexadecimal to ASCII), 252
IF-THEN, 210
IN_RANGE (within a range), 184
INC (increment), 189
INSERT (insert substring), 257
INV (invert), 227
JMP_LIST, 217
LED status, 282
LEFT (left substring), 255
LEN (length), 253
LIMIT, 192
LN (natural logarithm), 192
math, 187
math (floating-point), 192
MAX (maximum), 191
MC_ChangeDynamic, 365
MC_CommandTable, 362
MC_Halt, 352
MC_Home, 350
MC_MoveAbsolute, 354
MC_MoveJog, 360
MC_MoveRelative, 356
MC_MoveVelocity, 358
MC_Power, 346
MC_Reset, 349
MID (middle substring), 255
MIN (minimum), 191
MOD (modulo), 188
ModuleStates, 284
move, 194
MUX (multiplex), 229
N_TRIG, 169
NEG (negation), 189
NORM_X (normalize), 206
NOT OK, 185
OK, 185
operators, 147
OR, 226
OUT_RANGE (outside of a range), 184
overview, 145
P_TRIG, 169
PID overview, 324
PID_3STEP, 331
PID_3Step algorithm, 325, 331
PID_Compact, 328
PID_Compact algorithm, 325, 328
priority of operators, 147
program control, 209
program editor, 146
QRY_CINT (query cyclic interrupt), 278
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
REPEAT, 214
REPLACE (replace substring), 258
RETURN, 216
RIGHT (right substring), 255
ROL and ROR (rotate left and rotate right), 232
round, 204
run time meter, 237
S_CONV (value to string conversions), 241
S_MOV (move string), 240
SCALE_X (scale), 206
SEL (select), 228
Set and Reset, 166
SET_CINT (set cyclic interrupt), 276
SET_TIMEZONE (set time zone), 238

SHL and SHR (shift left and shift right), 231
SIN (sine), 192
SQR (square), 192
SQRT (square root), 192
SRT_DINT (start time delay interrupt), 279
STRG_VAL (string to value), 241
swap, 200
SWITCH, 218
T_ADD (add time), 234
T_COMBINE (combine times), 235
T_CONV (convert time), 233
T_DIFF (time difference), 234
T_SUB (subtract time), 234
TAN (tangent), 192
timer operations, 174
timers, 170
truncate, 204
UFILL_BLK (uninterruptible fill), 199
VAL_STRG (value to string), 241
Var section, 146
WHILE, 213
WR_SYS_T (write system time), 235
XOR (exclusive OR), 226
Security
access protection, 153
binding to a CPU or memory card, 156
copy protection, 156
CPU, 153
know-how protectionf for a code block, 154
lost password, 107
SEL (select), 228
Send message configuration, 552
Send parameters configuration, 118, 436, 465
SEND_CFG (send configuration), 535
SEND_PTP (send Point-to-Point data), 541
LENGH and BUFFER parameters, 543
Serial communication, 531
Service and support, 3
Set, 166
SET_CINT (set cyclic interrupt), 276
SET_TIMEZONE (set time zone), 238
Settings, 33
SGN_GET (get RS232 signals), 547
SGN_SET (set RS232 signals), 548
SHL and SHR (shift left and shift right), 231
Siemens security certificate, Web pages, 478, 490
Siemens technical support, 3
siemens_automation_language cookie, 523
Signal board (SB)
add modules, 112
analog output representation (current), 708, 733
analog output representation (voltage), 709, 734
configuration of parameters, 115
input representation (voltage), 669, 678, 687, 708, 733
installation, 46
overview, 22
power requirements, 761
removal, 46
SB 1221 DI 4, 200 kHz, 720
SB 1222 DQ, 200 kHz, 722
SB 1223 DI 2 / DQ 2, 726
SB 1223 DI 2 / DQ 2, 200 kHz, 724
SB 1231 AI 1 x 12 bit, 728
SB 1231 AI 1 x 16 bit RTD, 738
SB 1231 AI 1 x 16 bit Thermocouple, 734
SB 1232 AQ 1x12 bit, 731
step response times, 732
Signal handling errors, 547, 549
Signal module (SM)
add modules, 112
analog input representation (voltage), 669, 678, 687, 708, 733
analog output representation (current), 708, 733
analog output representation (voltage), 709, 734
configuration of parameters, 115
expansion cable, 51
installation, 47
overview, 22
power requirements, 761
removal, 48
SM 1221, 689
SM 1222, 691, 693
SM 1222 DQ8 RLY Changeover, 691
SM 1223, 696
SM 1223 specifications, 698
SM 1231 AI 4 x 13 bit, 700
SM 1231 AI 4 x 16 bit TC, 709
SM 1231 AI 4 x RTD x 16 bit, 714
SM 1231 AI 8 x 16 bit TC, 709
SM 1231 AI 8 x RTD x 16 bit, 714
SM 1232 AQ 2 x 14bit, 702
SM 1232 AQ 4 x 14bit, 702
SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit, 704
step response times, 707
Simulators, 759
SIN (sine), 192
Slave polling architecture, 562
SM 1231 RTD
selection tables, 718, 742
SM and SB
comparison chart, 20
device configuration, 109
SMS, 623

Software flow control, 551
Special characters
  User-defined Web pages, 505
Specifications
  analog input representation (voltage), 669, 678, 687, 708, 733
  analog output representation (current), 708, 733
  analog output representation (voltage), 709, 734
  ATEX approval, 658
  CB 1241 RS485, 753
  CE approval, 657
  CM 1241 RS232, 756
  CM 1241 RS485, 754
  CPU 1211C, 663
  CPU 1212C, 671
  CPU 1214C, 680
  C-Tick approval, 659
  cULus approval, 658
  electromagnetic compatibility (EMC), 660
  environmental conditions, 660
  FM approval, 658
  general technical specifications, 657
  industrial environments, 659
  input simulators, 759
  maritime approval, 659
  memory cards, 758
  protection, 661
  rated voltages, 661
  relay electrical service life, 663
  SB 1221 DI 4, 200 kHz, 720
  SB 1222 DQ 4, 200 kHz, 722
  SB 1223 DI 2 / DQ 2, 726
  SB 1223 DI 2 x / DQ 2, 200 kHz, 724
  SB 1231 AI 1 x 12 bit, 728
  SB 1231 AI 1 x 16 bit RTD, 738
  SB 1231 AI 1 x 16 bit RTD wiring diagram, 740
  SB 1231 AI 1 x 16 bit Thermocouple, 734
  SB 1231 AI 1 x 16 bit thermocouple wiring diagram, 738
  SB 1231 AI x 12 bit wiring diagram, 730
  SB 1232 AQ 1x12 bit, 731
  SM 1221 signal module, 689
  SM 1221 wiring diagram, 690
  SM 1222 DQ8 RLY Changeover, 691
  SM 1222 signal module, 691, 693
  SM 1222 wiring diagram, 693
  SM 1223 signal module, 696, 698
  SM 1223 wiring diagram, 697, 698
  SM 1231 AI 4 x 13 bit, 700
  SM 1231 AI 4 x 16 bit TC signal module, 709
  SM 1231 AI 4 x 16 bit TC wiring diagram, 711
  SM 1231 AI 4 x RTD x 16 bit signal module, 714
  SM 1231 AI 8 x 16 bit TC signal module, 709
  SM 1231 AI 8 x 16 bit TC wiring diagram, 711
  SM 1231 AI 8 x RTD x 16 bit signal module, 714
  SM 1231 RTD 4 x 16 bit wiring diagram, 716
  SM 1231 RTD 8 x 16 bit wiring diagram, 716
  SM 1232 AQ 2 x 14bit, 702
  SM 1232 AQ 4 x 14bit, 702
  SM 1234 AI 4 x 13 bit / AQ 2 x 14 bit, 704
  step response times (CPU), 669, 677, 686
  step response times (SB), 732
  step response times (SM), 707
  wiring diagrams SM 1231 analog input, 702
  wiring diagrams SM 1232 analog output, 704
  wiring diagrams SM 1234 analog input/output, 706
SQR (square), 192
SQRT (square root), 192
SRT_DINT (start time delay interrupt), 279
Standard Web pages, 473
  accessing from PC, 474
  changing operating mode, 479
  communication, 483
  cookie restrictions, 490
  Data Logs, 486
  Diagnostic, 480
  Identification, 480
  Intro, 478
  JavaScript restrictions, 489
  layout, 476
  logging in and out, 477
  Module information, 481
  secure access, 475
  Start, 479
  Variable Status, 484
Start conditions, 553
Start message character, 554
Start standard Web page, 479
Startup after POWER ON, 63
  startup processing, 65
STARTUP mode
  force operation, 648
Startup OB, 67
Startup parameters, 102
Status
  LED indicators, 633
  LED indicators (communication interface), 529
  LED instruction, 282
STEP 7
  add modules, 112
  add new device, 110
  Adding a PROFINET IO device, 437
  adding inputs or outputs to a LAD or FBD instruction, 31

AS-i, 454
AS-i port, 454
assigning an IP address to an online CPU, 125
block calls, 61
calling code blocks within the user program, 138
capturing the status of a code block, 34
capturing values of a DB, 644
changing the settings, 33
communication load, 74
comparing and synchronizing, 641
configuring the CPU, 114
configuring the modules, 115
copying blocks from an online CPU, 157
counters, 178
cycle time, 74
cycle time, 74
data block (DB), 61
device configuration, 109
download, 157
drag and drop between editors, 33
Ethernet port, 127
expandable inputs or outputs, 32
favorites, 29
force, 647
force operation, 648
function (FC), 140
function block (FB), 61, 140
HSC configuration, 323
initial value of an FB, 140
inserting instructions, 29
instance data block (DB), 140
linear and structured programs, 136
memory card, 99, 758
monitoring, 642, 643
network connection, 117
operating modes, 63
operation, 645
operator panel, 34
password protection, 154
Portal view, 28
priority class (OB), 66
PROFIBUS, 449
PROFIBUS port, 449
PROFINET, 127
program card, 99
Project view, 28
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
resetting the start values of a DB, 644
restoring the status of a code block, 34
run time meter, 237
RUN/STOP buttons, 34
  startup processing, 65
  time synchronization property (PROFINET), 132
  transfer card, 99
  types of code blocks, 61
  unplugged modules, 38
  valid FC, FB, and DB numbers, 61
  WR_SYS_T (write system time), 235
STEP 7 programming
  PtP example program, 565
  user-defined Web pages, 508
STEP 7 web pages, 4
Stop bits, 550
STOP mode, 63, 639
  enable outputs in STOP mode, 647
  force operation, 648
  operator panel, 34
  toolbar buttons, 34
STP (stop PLC scan cycle), 222
STRG_VAL (string to value), 241
String
  S_MOVE (string move), 240
  string data overview, 240
  String data type, 91
  string operations overview, 253
Structured programming, 136, 138
  calling blocks, 138
SUB (subtract), 187
Subnet mask, 127
Support, 3
SWAP, 200
SWITCH, 218
Switching languages, user-defined Web pages, 522
Synchronization
  time synchronization property (PROFINET), 132
System clock
  RD_LOC_T (read local time), 235
  RD_SYS_T (read system time), 235
  WR_SYS_T (write system time), 235
System memory byte, 79
System requirements, 27

T

T_ADD (add time), 234
T_COMBINE (combine times), 235
T_CONFIG, 421
T_CONV (convert time), 233
T_DIFF (time difference), 234
T_SUB (subtract timet), 234
Tags
  force, 647
  force operation, 648

monitor, 642
status, 642
TAN (tangent), 192
Task cards
columns and headers, 32, 599
TCON, 408
configuration, 118
connection IDs, 396
connection parameters, 120
TCON_Param, 120
TCP
ad hoc mode, 400
connection configuration, 118
connection IDs, 396
parameters, 120
protocol, 399
TCP/IP communication, 399
TDISCON, 408
Technical specifications, 657
Technical support, 3
Technological objects
HSC (high-speed counter), 317
Telecontrol, 620
Teleservice communication
TM_MAIL, 627
TeleService via GPRS, 620, 625
Terminal block connector, 50
Terminal emulator for PtP example program, 566
Testing the program, 159
Thermal zone, 39, 42
Thermocouple
basic operation, 712, 736
cold junction compensation, 712, 736
SB 1231 Thermocouple filter selection table, 736
SB 1231 Thermocouple selection table, 736
SM 1231 Thermocouple filter selection table, 712
SM 1231 Thermocouple selection table, 712
TIA Portal
Portal view, 28
Project view, 28
Time
DTL (date and time long data type), 90
RD_LOC_T (read local time), 235
RD_SYS_T (read system time), 235
SET_TIMEZONE (set time zone), 238
T_ADD (add time), 234
T_COMBINE (combine times), 235
T_CONV (convert time), 233
T_DIFF (time difference), 234
T_SUB (subtract time), 234
Time data type, 89
TOD (time of day data type), 89
WR_SYS_T (write system time), 235
Time delay interrupt), 279
Time of day
configuring the online CPU, 638
Time synchronization property, 132
Time-error interrupt OB, 67
Timers
operation, 174
quantity, 19, 665, 673, 682
RT (reset timer), 170
size, 19, 665, 673, 682
TOF (off-delay timer), 170
TON (on-delay delay timer), 170
TONR (on-delay retentive) timer, 170
TP (pulse delay timer), 170
TM_MAIL, 627
Transfer card, 102
configure the startup parameters, 102
empty transfer card for a lost password, 107
inserting into CPU, 100
lost password, 107
operation, 99
order number, 758
overview, 99
Transmission block (T-block), 434
Transmit configuration errors, 536
Transmit message configuration, 552
PtP example program, 563
Transmit runtime errors, 543
TRCV, 408
ad hoc mode, 400
connection IDs, 396
TRCV_C, 401, 436
ad hoc mode, 400
configuration, 118
connection IDs, 396
connection parameters, 120
TRCV_C instruction configuration, 437
Triggering values in the watch table, 646
Troubleshooting
diagnostics buffer, 640
LED indicators, 633
TRUNC (truncate), 204
TS Adapter, 20
installing a TS module, 53
installing on a DIN rail, 54
installing on a wall, 56
SIM card, 53
TSAP, 399
TSAP (transport service access points), 120, 401, 434, 464
TSAPs

restricted, 429
TSEND, 408
connection IDs, 396
TSEND_C, 435
configuration, 118
connection IDs, 396
connection parameters, 120
TSEND_C, 435
TSEND_C instruction configuration, 436
TURCV, 416
configuration, 118
connection parameters, 120
TUSEND, 416
configuration, 118
parameters, 120

U

UDP
connection configuration, 118
parameters, 120
UDP protocol, 399
UFILL_BLK (uninterruptible fill), 199
Uninterruptible move (UMOVE_BLK), 194
Unplugged modules, 38
Unspecific CPU, 111
Updating user-defined Web pages, 494
Uploading
copying blocks from an online CPU, 157
user program, 157
User interface
Portal view, 28
Project view, 28
User program
adding inputs or outputs to LAD or FBD instructions, 31
binding to a CPU or memory card, 156
calling code blocks within the user program, 138
copying blocks from an online CPU, 157
download, 157
drag and drop between editors, 33
expandable instructions, 32
favorites, 29
inserting instructions, 29
linear and structured programs, 136
memory card, 99
organization block (OB), 138
password protection, 154
program card, 99
transfer card, 99
User-defined Web pages, 473, 492
accessing from PC, 510
activating and deactivating from control DB, 526
AWP commands for accessing S7-1200 data, 494
configuring, 507
creating fragments, 503
creating with HTML editor, 493
deleting program blocks, 508
downloading corresponding DBs, 509
enabling with WWW instruction, 508
example, 511
generating program blocks, 508
handling special characters, 505
HTML listing, 516
importing fragments, 504
load memory constraints, 510
manual fragment DB control, 526
multiple language configuration, 525
multiple languages, 522
programming in STEP 7, 508
reading special variables, 497
reading variables, 495
refreshing, 494
writing special variables, 499
writing variables, 496
USS protocol library
general drive setup inforamtion, 578
overview, 567
requirements for using, 568
status codes, 576
USS_DRV, 570
USS_PORT, 573
USS_RPM, 574
USS_WPM, 575

V

VAL_STRG (value to string), 241
Variable index for an array, 198
Variable Status standard Web page, 484
Variables
monitoring and modifying from PC, 484
Versions of instructions, 32, 599
Visualization
HMI devices, 25

W

Wait time, 550
Warm restart, 63
Watch table
enable outputs in STOP mode, 647
force, 159

memory card does not contain force values, 99
monitor, 642
operation, 645
trigger values, 646
Watchdog, 221
Web pages
STEP 7, 4
Web server, 473
constraints, 488
enabling, 474
Quotation mark conventions, 505
standard Web pages, 474
update rate, 474
Web server, user-defined Web pages, 492
Wiring diagrams
CPU 1211C, 671
CPU 1212C, 680
CPU 1214C, 688
SB 1221 DI 4, 200 kHz, 721
SB 1222 DQ 4, 200 kHz, 723
SB 1223 DI 2 / DQ 2, 200 kHz, 726
SB 1223 DI 2/ DQ 2, 728
SB 1231 AI 1 x 16 bit RTD, 740
SB 1231 AI 1 x 16 bit thermocouple, 738
SB 1231 AI x 12 bit, 730
SB 1232 AQ 1 x 12 bit, 731
SM 1221 signal module, 690
SM 1222 signal module, 693
SM 1223 signal module, 697, 698
SM 1231 AI 4 x 16 bit TC, 711
SM 1231 AI 8 x 16 bit TC, 711
SM 1231 analog input, 702
SM 1231 RTD 4 x 16 bit, 716
SM 1231 RTD 8 x 16 bit, 716
SM 1232 analog output, 704
SM 1234 analog input/output, 706
Wiring guidelines, 58
clearance for airflow and cooling, 39
grounding, 58
prerequisites, 56
Work memory, 18
CPU 1211C, 663
CPU 1212C, 671
CPU 1214C, 680
WR_SYS_T (write system time), 235
WRIT_DBL, 310
WRREC, 260, 265
WWW (enable user-defined Web pages), 508

X

XON / XOFF, 551
XOR (exclusive OR), 226