SIEMENS

SIMATIC

S7
S7-1200 Programmable controller

System Manual

Preface

Product overview 1

Installation 2

PLC concepts 3

Device configuration 4

Programming concepts 5

Programming instructions 6

PROFINET 7

Point-to-Point (PtP) communications 8

Online and diagnostic tools 9

Technical specifications A

Calculating a power budget B

Order numbers C

11/2009
A5E02486680-02

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 corresponding 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 for the specific task, 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 adhered to. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of the 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

Ordernumber: 6ES7298-8FA30-8BH0
Ⓟ 10/2009

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 is valid for STEP 7 Basic V10.5 and the S7-1200 product family. For a complete list of the S7-1200 products described in this manual, refer to the technical specifications (Page 279).

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

Refer to the technical specifications (Page 279) for more information.

Service and support

In addition to our documentation, we offer our technical expertise on the Internet at:

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.

Table of contents

Preface 3

1 Product overview 11

1.1 Introducing the S7-1200 PLC 11

1.2 Signal boards 13

1.3 Signal modules 13

1.4 Communication modules 14

1.5 STEP 7 Basic 14
1.5.1 Different views to make the work easier 15
1.5.2 Help when you need it 16

1.6 Display panels 19

2 Installation 21

2.1 Installation and removal procedures 24
2.1.1 Installing and removing the CPU 26
2.1.2 Installing and removing a signal module 27
2.1.3 Installing and removing a communication module 29
2.1.4 Installing and removing a signal board 30
2.1.5 Removing and reinstalling the S7-1200 terminal block connector 31

2.2 Wiring guidelines 32

3 PLC concepts 37

3.1 Execution of the user program 37
3.1.1 Operating modes of the CPU 39
3.1.2 Event execution priorities and queuing 43
3.1.3 CPU memory 48
3.1.4 Password protection for the S7-1200 CPU 52
3.1.5 Recovery from a lost password 53

3.2 Data storage, memory areas and addressing 53

3.3 Data types 57

3.4 Using a memory card 61
3.4.1 Inserting a memory card in the CPU 62
3.4.2 Configuring the startup parameter of the CPU before copying the project to the memory card 63
3.4.3 Transfer card 63
3.4.4 Program card 65

4 **Device configuration** **69**
4.1 Inserting a CPU 70
4.2 Detecting the configuration for an unspecified CPU 71
4.3 Configuring the operation of the CPU 72
4.4 Adding modules to the configuration 73
4.5 Configuring the parameters of the modules 74
4.6 Creating a network connection 75
4.7 Configuring an IP address in your project 76
5 **Programming concepts** **79**
5.1 Guidelines for designing a PLC system 79
5.2 Structuring your user program 80
5.3 Using blocks to structure your program 81
5.3.1 Organization block (OB) 82
5.3.2 Function (FC) 84
5.3.3 Function block (FB) 84
5.3.4 Data block (DB) 86
5.4 Understanding data consistency 86
5.5 Selecting the programming language 87
5.6 Copy protection 89
5.7 Downloading the elements of your program 89
5.8 Uploading the elements of your program 90
5.9 Debugging and testing the program 91
6 **Programming instructions** **93**
6.1 Basic instructions 93
6.1.1 Bit logic 93
6.1.1.1 Set and reset instructions 95
6.1.1.2 Positive and negative edge instructions 97
6.1.2 Timers 99
6.1.3 Counters 102
6.1.3.1 Counters 102
6.1.3.2 CTRL_HSC instruction 105
6.1.3.3 Operation of the high-speed counter 107
6.1.3.4 Configuration of the HSC 110
6.1.4 Compare 111
6.1.5 Math 113
6.1.5.1 MOD instruction 114
6.1.6 Move 119
6.1.6.1 Swap instruction 122
6.1.7 Convert 123
6.1.7.1 Scale and normalize instructions 124
6.1.8 Program control 126
6.1.9 Logical operations 127
6.1.10 Shift and Rotate 130

6.2 Extended instructions....132
6.2.1 Common error parameters for extended instructions....132
6.2.2 Clock and calendar instructions....132
6.2.3 String and character instructions....136
6.2.3.1 String data overview....136
6.2.3.2 String conversion instructions....136
6.2.3.3 String operation instructions....144
6.2.4 Program control instructions....150
6.2.4.1 Reset scan cycle watchdog instruction....150
6.2.4.2 Stop scan cycle instruction....151
6.2.4.3 Get Error instructions....151
6.2.5 Communications instructions....154
6.2.5.1 Open Ethernet Communication....154
6.2.5.2 Point-to-Point instructions....167
6.2.6 Interrupt instructions....167
6.2.6.1 Attach and detach instructions....167
6.2.6.2 Start and cancel time delay interrupt instructions....170
6.2.6.3 Disable and Enable alarm interrupt instructions....172
6.2.7 PID control....172
6.2.8 Motion control instructions....173
6.2.9 Pulse instruction....174
6.2.9.1 CTRL_PWM instruction....174

6.3 Global library instructions....177
6.3.1 USS....177
6.3.1.1 Requirements for using the USS protocol....177
6.3.1.2 USS_DRV instruction....180
6.3.1.3 USS_PORT instruction....182
6.3.1.4 USS_RPM instruction....183
6.3.1.5 USS_WPM instruction....184
6.3.1.6 USS status codes....186
6.3.2 MODBUS....187
6.3.2.1 MB_COMM_LOAD....187
6.3.2.2 MB_MASTER....189
6.3.2.3 MB_SLAVE....199

7 PROFINET....209

7.1 Communication with a programming device....210
7.1.1 Establishing the hardware communications connection....211
7.1.2 Configuring the devices....211
7.1.3 Assigning Internet Protocol (IP) addresses....212
7.1.3.1 Assigning IP addresses to programming and network devices....212
7.1.3.2 Assigning an IP address online....214
7.1.3.3 Configuring an IP address in your project....217
7.1.4 Testing the PROFINET network....219

7.2 HMI-to-PLC communication....221
7.2.1 Configuring the logical network connections between an HMI and a CPU....222

7.3 PLC-to-PLC communication....223
7.3.1 Configuring the logical network connections between two CPUs....224
7.3.2 Configuring transmit (send) and receive parameters....225
7.3.2.1 Configuring the TSEND_C instruction transmit (send) parameters....225
7.3.2.2 Configuring the TRCV_C instruction receive parameters....229

7.4 Reference Information ... 233
7.4.1 Locating the Ethernet (MAC) address on the CPU ... 233
7.4.2 Configuring Network Time Protocol synchronization ... 234

8 Point-to-Point (PtP) communications ... 237

8.1 Using the RS232 and RS485 communication modules ... 237

8.2 Configuring the communication ports ... 238

8.3 Managing flow control ... 239

8.4 Configuring the transmit (send) and receive parameters ... 241

8.5 Programming the PtP communications ... 246
8.5.1 Polling architecture ... 247

8.6 Point-to-Point instructions ... 248
8.6.1 Common parameters for Point-to-Point instructions ... 248
8.6.2 PORT_CFG instruction ... 250
8.6.3 SEND_CFG instruction ... 251
8.6.4 RCV_CFG instruction ... 253
8.6.5 SEND_PTP instruction ... 259
8.6.6 RCV_PTP instruction ... 262
8.6.7 RCV_RST instruction ... 263
8.6.8 SGN_GET instruction ... 264
8.6.9 SGN_SET instruction ... 265

8.7 Errors ... 266

9 Online and diagnostic tools ... 269

9.1 Status LEDs ... 269

9.2 Going online and connecting to a CPU ... 270

9.3 Setting the IP address and time of day ... 271

9.4 CPU operator panel for the online CPU ... 272

9.5 Monitoring the cycle time and memory usage ... 272

9.6 Displaying diagnostic events in the CPU ... 272

9.7 Watch tables for monitoring the user program ... 273

A Technical specifications ... 279

A.1 General Technical Specifications ... 279

A.2 CPUs ... 284
A.2.1 CPU 1211C Specifications ... 284
A.2.2 CPU 1212C Specifications ... 289
A.2.3 CPU 1214C Specifications ... 294

A.3 Digital signal modules (SMs) ... 299
A.3.1 SM 1221 Digital Input Specifications ... 299
A.3.2 SM 1222 Digital Output Specifications ... 301
A.3.3 SM 1223 Digital Input/Output Specifications ... 303

A.4 Analog signal modules (SMs) ... 306
A.4.1 SM 1231, SM 1232, SM 1234 Analog Specifications ... 306

A.5 Signal boards (SBs) ... 313
A.5.1 SB 1223 2 X 24 VDC Input / 2 X 24 VDC Output Specifications ... 313
A.5.2 SB 1232 1 Analog Output Specifications ... 315

A.6 Communication modules (CMs)....317
A.6.1 CM 1241 RS485 Specifications....317
A.6.2 CM 1241 RS232 Specifications....318
A.7 SIMATIC memory cards....318
A.8 Input simulators....319
A.9 I/O expansion cable....320
B Calculating a power budget....321
B.1 Calculating a sample power requirement....322
B.2 Calculating your power requirement....323
C Order numbers....325
Index....329

Product overview 1

1.1 Introducing the S7-1200 PLC

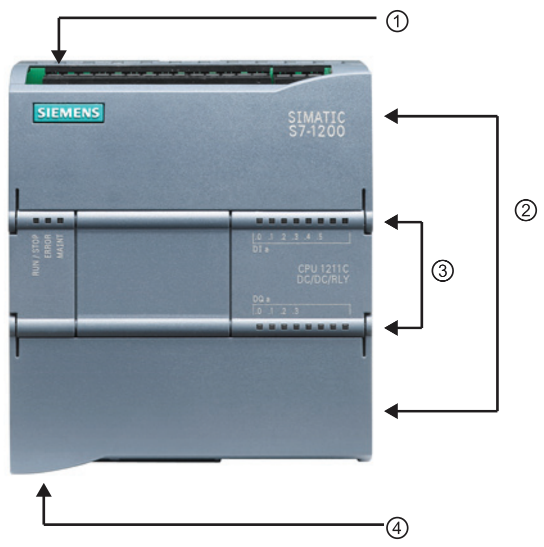
The S7-1200 programmable logic controller (PLC) provides the flexibility and power to control a wide variety of devices in support of your automation needs. The compact design, flexible configuration, and powerful instruction set combine to make the S7-1200 a perfect solution for controlling a wide variety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

The CPU combines a microprocessor, an integrated power supply, input circuits, and output circuits in a compact housing to create a powerful PLC. 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.

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

- Every CPU provides password protection that allows you to configure access to the CPU functions.
- You can use "know-how protection" to hide the code within a specific block. See the "Programming concepts" (Page 89) chapter for details.

The CPU provides a PROFINET port for communication over a PROFINET network. Communication modules are available for communicating over RS485 or RS232 networks.



① Power connector

② Removable user wiring connectors (behind the doors)

② Memory card slot under top door

③ Status LEDs for the on-board I/O

④ PROFINET connector (on the bottom of the CPU)

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 279).

Feature	CPU 1211C	CPU 1212C	CPU 1214C
Physical size (mm)	90 x 100 x 75		110 x 100 x 75
User memory • Work memory • Load memory • Retentive memory	 • 25 Kbytes • 1 Mbyte • 2 Kbytes		 • 50 Kbytes • 2 Mbytes • 2 Kbytes
Local on-board I/O • Digital • Analog	 • 6 inputs/4 outputs • 2 inputs	 • 8 inputs/6 outputs • 2 inputs	 • 14 inputs/10 outputs • 2 inputs
Process image size	1024 bytes inputs (I) and 1024 bytes outputs (Q)		
Bit memory (M)	4096 bytes		8192 bytes
Signal modules expansion	None	2	8
Signal board	1		
Communication modules	3 (left-side expansion)		
High-speed counters • Single phase • Quadrature phase	3 • 3 at 100 kHz • 3 at 80 kHZ	4 • 3 at 100 kHz 1 at 30 kHz • 3 at 80 kHz 1 at 20 kHz	6 • 3 at 100 kHz 3 at 30 kHz • 3 at 80 kHz 3 at 20 kHz
Pulse outputs	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		

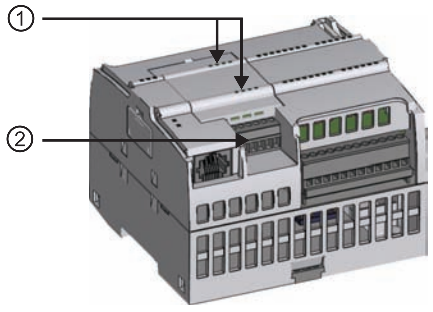
The S7-1200 family provides a variety of signal modules and signal boards for expanding the capabilities of the CPU. You can also install additional communication modules to support other communication protocols. For detailed information about a specific module, see the technical specifications (Page 279).

Module		Input only	Output only	Combination in/out
Signal module (SM)	Digital	8 x DC In	8 x DC Out 8 x Relay Out	8 x DC In/8 x DC Out 8 x DC In/8 x Relay Out
		16 x DC In	16 x DC Out 16 x Relay Out	16 x DC In/16 x DC Out 16 x DC In/16 x Relay Out
	Analog	4 x Analog In 8 x Analog In	2 x Analog Out 4 x Analog Out	4 x Analog In/2 x Analog Out
Signal board (SB)	Digital	-	-	2 x DC In/2 x DC Out
	Analog	-	1 x Analog Out	-
Communication module (CM) • RS485 • RS232				

1.2 Signal boards

A signal board (SB) allows you to add I/O to your CPU. You can add one SB with either digital or analog I/O. An SB connects on the front of the CPU.

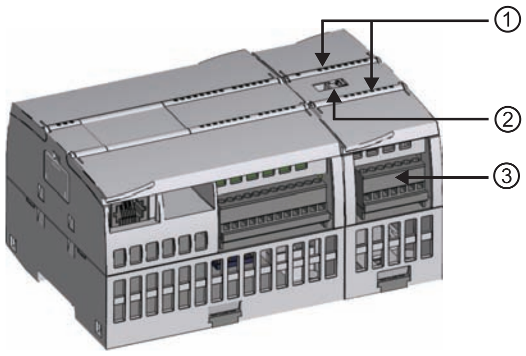
- SB with 4 digital I/O (2 x DC inputs and 2 x DC outputs)
- SB with 1 analog output



①	Status LEDs on the SB
②	Removable user wiring connector

1.3 Signal modules

You can use signal modules to add additional functionality to the CPU. Signal modules connect to the right side of the CPU.

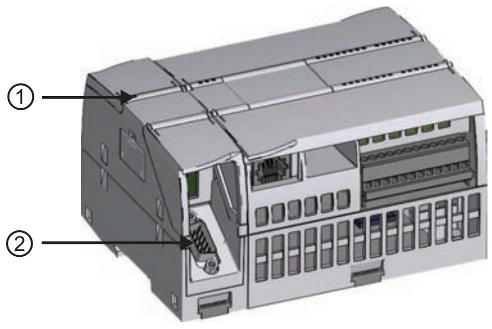


①	Status LEDs for the I/O of the signal module
②	Bus connector
③	Removable user wiring connector

1.4 Communication modules

The S7-1200 family provides communication modules (CMs) for additional functionality to the system. There are two communication modules: RS232 and RS485.

- The CPU supports up to 3 communication modules
- Each CM connects to the left side of the CPU (or to the left side of another CM)



①	Status LEDs for the communication module
②	Communication connector

1.5 STEP 7 Basic

The STEP 7 Basic software 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 PLCs and HMI devices. STEP 7 Basic provides two programming languages (LAD and FBD) for convenience and efficiency in developing the control program for your application, and also provides the tools for creating and configuring the HMI devices in your project.

To help you find the information you need, STEP 7 Basic provides an extensive online help system.

To install STEP 7 Basic, insert the CD into the CD-ROM drive of your computer. The installation wizard starts automatically and prompts you through the installation process. Refer to the Readme file for more information.

Note

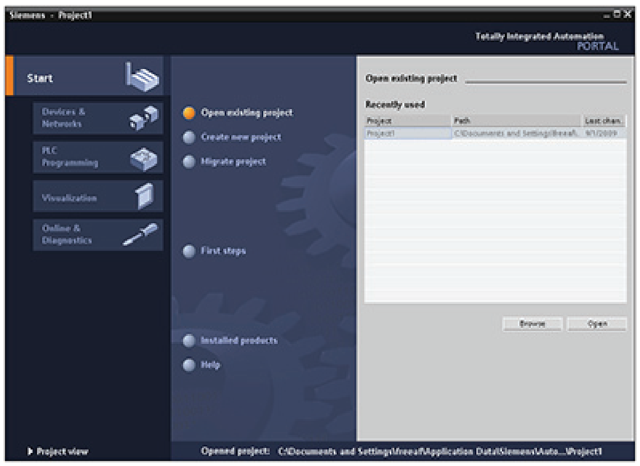
To install the STEP 7 Basic software on a PC running Windows 2000, Windows XP, or Windows Vista operating system, you must log in with Administrator privileges.

1.5.1 Different views to make the work easier

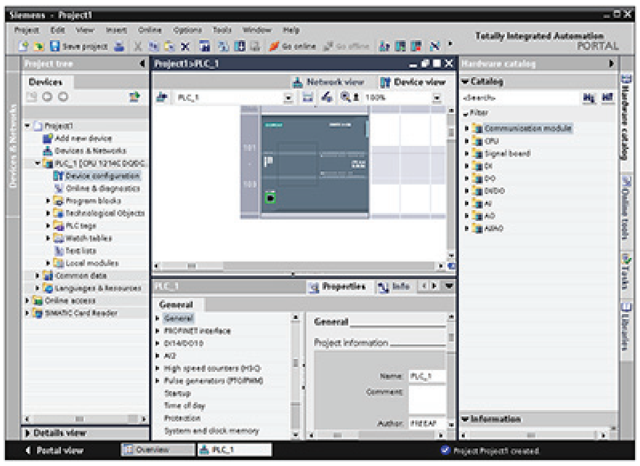
To help increase your productivity, the Totally Integrated Automation Portal provides two different views of the toolset: a task-oriented set of portals that are organized on the functionality of the tools (Portal view), or a project-oriented view of the elements within the project (Project view). Choose which view helps you work most efficiently. With a single click, you can toggle between the Portal view and the Project view.

The Portal view provides a functional view of the project tasks and organizes the functions of the tools according to the tasks to be accomplished, such as creating the configuration of the hardware components and networks.

You can easily determine how to proceed and which task to choose.



The Project view provides access to all of the components within a project. With all of these components in one place, you have easy access to every aspect of your project. The project contains all of the elements that have been created or completed.



1.5.2 Help when you need it

Finding answers to your questions quickly

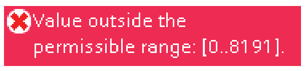
To help you resolve issues quickly and efficiently, STEP 7 Basic provides intelligent point-of-need assistance:

- An entry field provides "rollout" help to assist you with entering the correct information (valid ranges and type of data) for that field. For example, if you were to enter an invalid value, a message text box would roll out to provide the range of valid values.
- Some of the tool tips in the interface (such as for the instructions) "cascade" to provide additional information. Some of the cascading tool tips link to specific topics in the online information system (online help).

In addition, STEP 7 Basic has a comprehensive information system that fully describes the functionality of the SIMATIC tools.

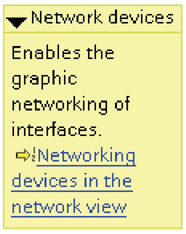
Rollout help and cascading tool tips

Entry fields of various dialogs and task cards provide feedback in the form of a message box that rolls out and informs you about the range or types of data required.


Value outside the permissible range: [0..8191].

The elements of the software interface provide tool tips to explain the functionality of the element. Some of the elements, such as the "Open" or "Save" icons, require no additional information. However, some of the elements provide a mechanism for displaying additional description about the element. This additional information "cascades" in a box from the tool tip. (A black triangle alongside the tool tip signifies that more information is available.)

Hovering over an element of the software interface displays the tool tip. To display additional information, simply hover your cursor over the tool tip. Some of the cascading tool tips also provide links to related topics in the information system. Clicking the link displays the specific topic.


Network devices
Enables the graphic networking of interfaces.
Networking devices in the network view

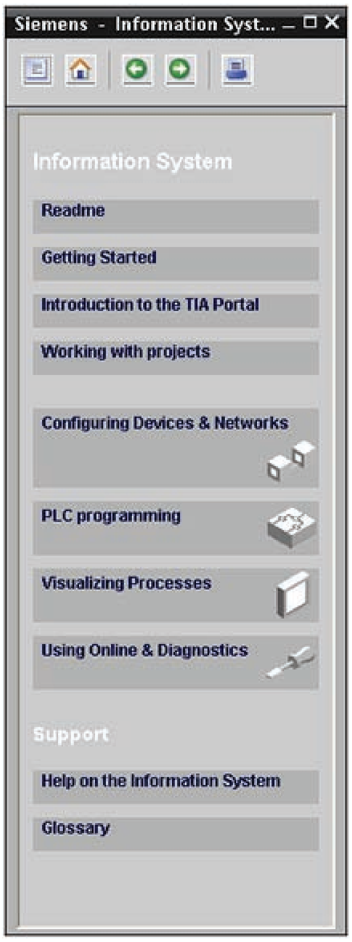
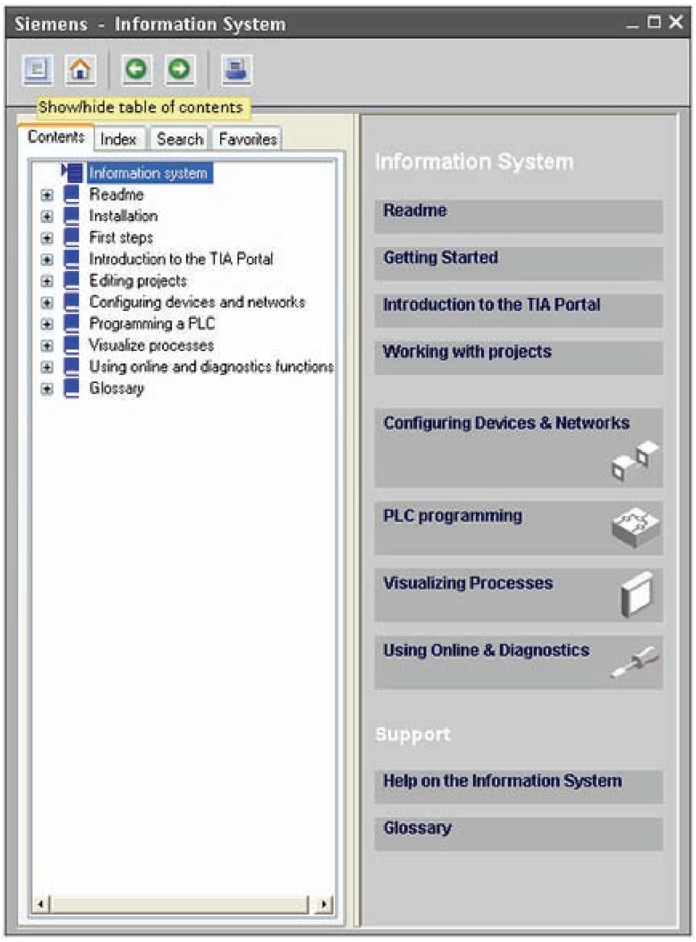
Information system

STEP 7 Basic provides a comprehensive online information and help system that describes all of the SIMATIC products that you have installed. The information system also includes reference information and examples. To display the information system, choose from the following access points:

- From the Portal view, select the Start portal and click the "Help" command.
- From the Project view, select the "Show help" command in the "Help" menu.
- From a cascading tool tip, click a link to display more information about that topic.

The information system opens in a window that does not obscure the work areas.

Click the "Show/hide contents" button on the information system to display the contents and undock the help window. You can then resize the help window. Use the "Contents" or "Index" tabs to search through the information system by topic or by key word.

Help window (default)	Help window with contents displayed
	

Siemens - Information Syst...
Information System
Readme
Getting Started
Introduction to the TIA Portal
Working with projects
Configuring Devices & Networks
PLC programming
Visualizing Processes
Using Online & Diagnostics
Support
Help on the Information System
Glossary

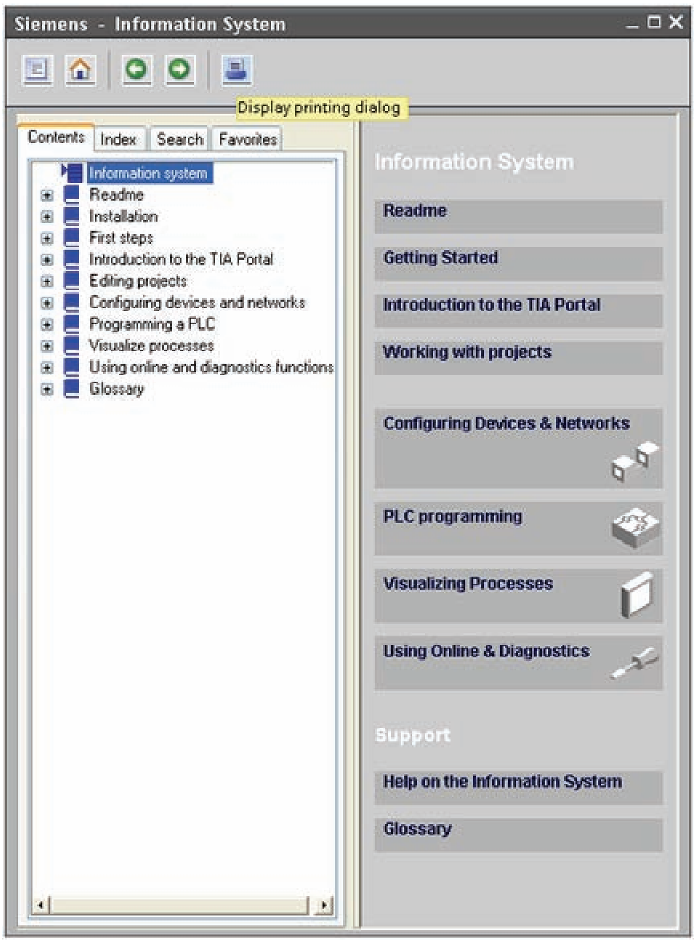
Siemens - Information System
Show/hide table of contents
Contents
Index
Search
Favorites
Information system
Readme
Installation
First steps
Introduction to the TIA Portal
Editing projects
Configuring devices and networks
Programming a PLC
Visualize processes
Using online and diagnostics functions
Glossary
Information System
Readme
Getting Started
Introduction to the TIA Portal
Working with projects
Configuring Devices & Networks
PLC programming
Visualizing Processes
Using Online & Diagnostics
Support
Help on the Information System
Glossary

Note

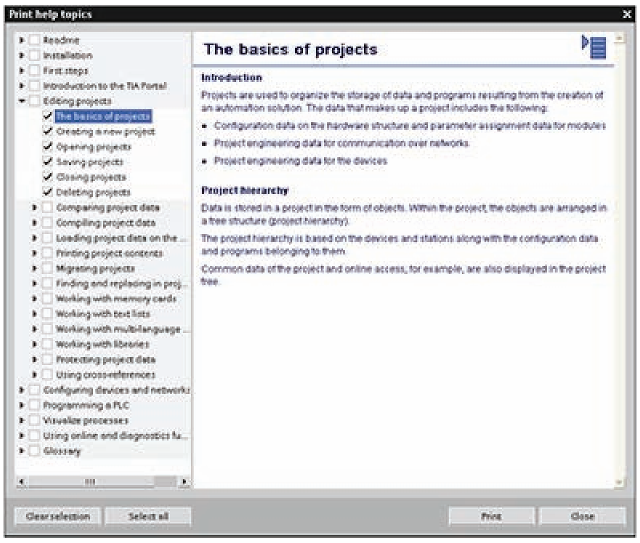
If STEP 7 Basic is maximized, clicking the "Show/hide contents" button does not undock the help window. Click the "Restore down" button to undock the help window. You can then move and resize the help window.

Printing topics from the information system

To print from the information system, click the "Print" button on the help window.



To print from the information system, click the "Print" button on the help window.



The basics of projects

Introduction

Projects are used to organize the storage of data and programs resulting from the creation of an automation solution. The data that makes up a project includes the following:

- Configuration data on the hardware structure and parameter assignment data for modules
- Project engineering data for communication over networks
- Project engineering data for the devices

Project hierarchy

Data is stored in a project in the form of objects. Within the project, the objects are arranged in a tree structure (project hierarchy).

The project hierarchy is based on the devices and stations along with the configuration data and programs belonging to them.

Common data of the project and online access, for example, are also displayed in the project tree.

The "Print" dialog allows you to select the topics to print. Make certain that the panel displays a topic. You can then select any other topic to print. Click the "Print" button to send the selected topics to your printer.

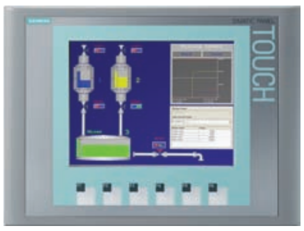
1.6 Display panels

As visualization becomes 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 are have a protection rating for IP65 and have CE, UL, cULus, and NEMA 4x certification.

KTP 400 Basic PN

- Mono (STN, gray scale)
- 4" touch screen with 4 tactile keys
- Portrait or landscape
- Size: 3.8"
- Resolution: 320 x 240

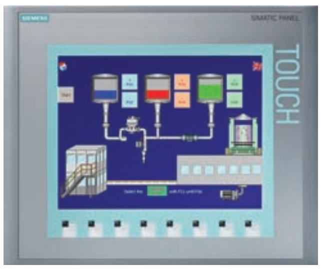
- 128 tags
- 50 process screens
- 200 alarms
- 25 curves
- 32 KB recipe memory
- 5 recipes, 20 data records, 20 entries



KTP 600 Basic PN

- Color (TFT, 256 colors) or Mono (STN, gray scales)
- 6" touch screen with 6 tactile keys
- Portrait or landscape
- Size: 5.7"
- Resolution: 320 x 240

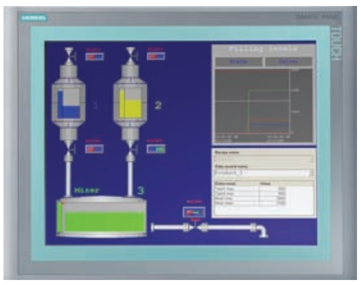
- 128 tags
- 50 process screens
- 200 alarms
- 25 curves
- 32 KB recipe memory
- 5 recipes, 20 data records, 20 entries



KTP1000 Basic PN

- Color (TFT, 256 colors)
- 10" touch screen with 8 tactile keys
- Size: 10.4"
- Resolution: 640 x 480

- 256 tags
- 50 process screens
- 200 alarms
- 25 curves
- 32 KB recipe memory
- 5 recipes, 20 data records, 20 entries



TP1500 Basic PN

- Color (TFT, 256 colors)
- 15" touch screen
- Size: 15.1"
- Resolution: 1024 x 768

- 256 tags
- 50 process screens
- 200 alarms
- 25 curves
- 32 KB recipe memory (integrated flash)
- 5 recipes, 20 data records, 20 entries

Installation 2

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.

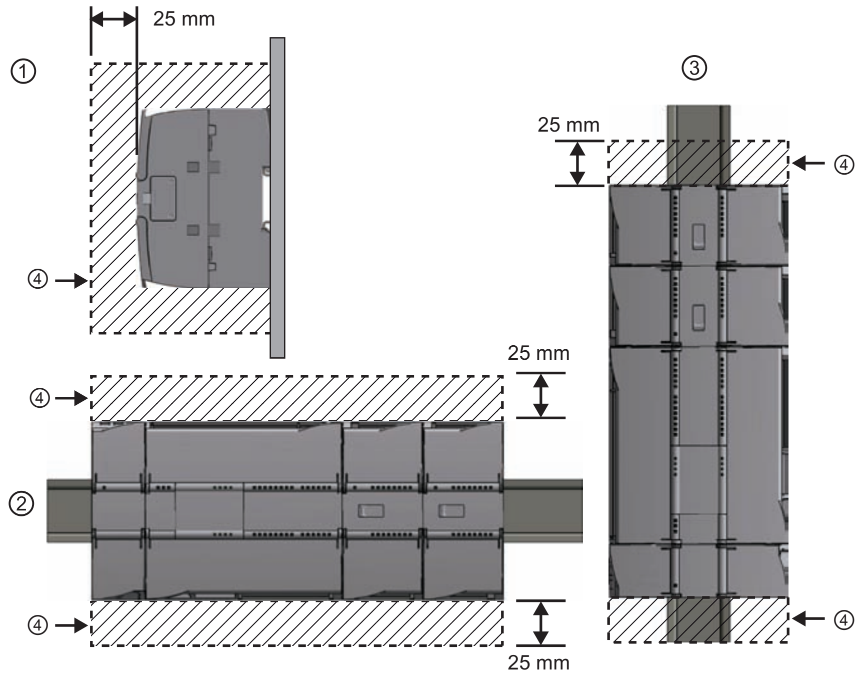
Provide adequate clearance for cooling and wiring

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 so that the CPU is at the low end of the assembly.

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



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

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 279) for information about the 5 VDC logic budget supplied by your CPU and the 5 VDC power requirements of the signal modules, signal board, and communication modules. Refer to the "Calculating a power budget" (Page 321) 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 279) for the 24 VDC sensor supply power budget for your particular S7-1200 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.

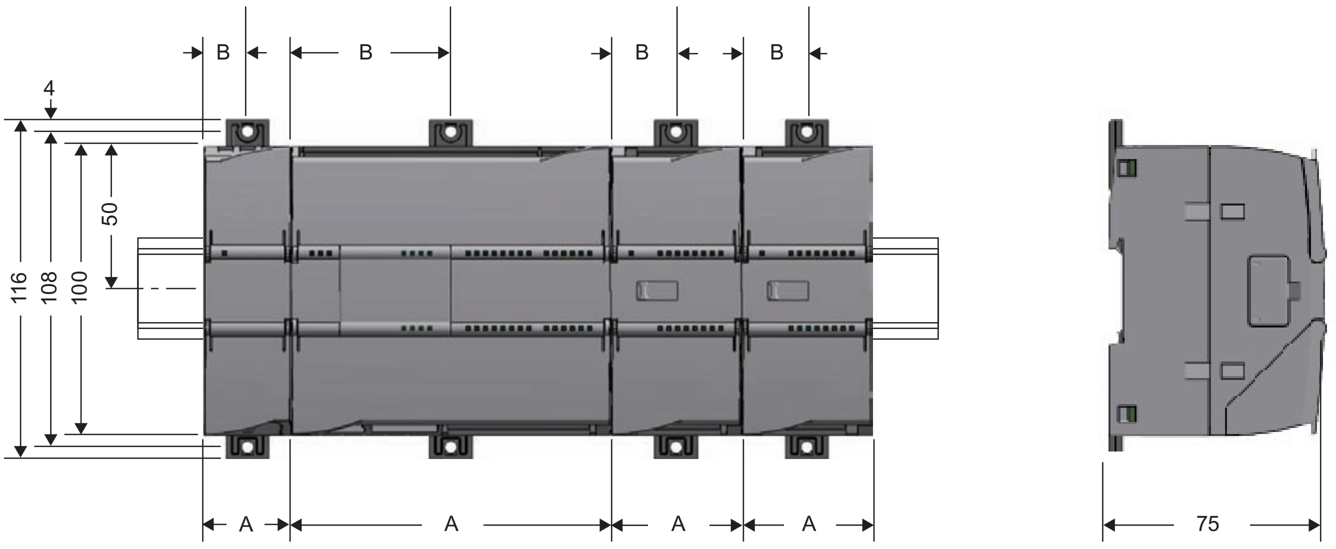
⚠ WARNING
Connecting an external 24 VDC power supply in parallel with the 24 VDC sensor supply can result in a conflict between the two supplies as each seeks to establish its own preferred output voltage level. The result of this conflict can be shortened lifetime or immediate failure of one or both 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.

2.2 Installation and removal procedures

Mounting dimensions (mm)



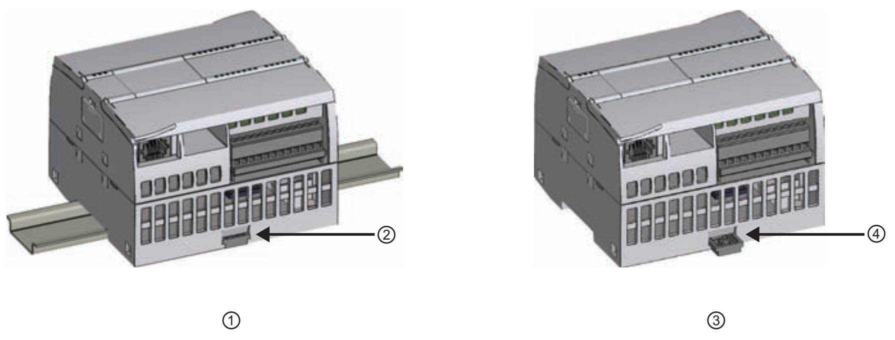
S7-1200 Devices		Width A	Width B
CPUs:	CPU 1211C and CPU 1212C	90 mm	45 mm
	CPU 1214C	110 mm	55 mm
Signal modules:	8 and 16 point DC and Relay (8I, 16I, 8Q, 16Q, 8I/8Q) Analog (4AI, 8AI, 4AI/4AQ, 2AQ, 4AQ)	45 mm	22.5 mm
	16I/16Q Relay (16I/16Q)	70 mm	35 mm
Communication modules:	CM 1241 RS232 and CM 1241 RS485	30 mm	15 mm

The CPUs, SMs and CMs support DIN rail mounting and panel mounting. 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	③	Panel installation
②	DIN rail clip in latched position	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.

2.2.1 Installing and removing the CPU

Installation

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.

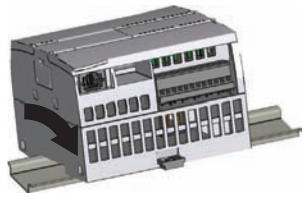
To mount the CPU on a panel, follow these steps:

1. Locate, drill, and tap the mounting holes (M4 or American Standard number 8), using the dimensions shown in the mounting dimensions.
2. 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.
3. 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.

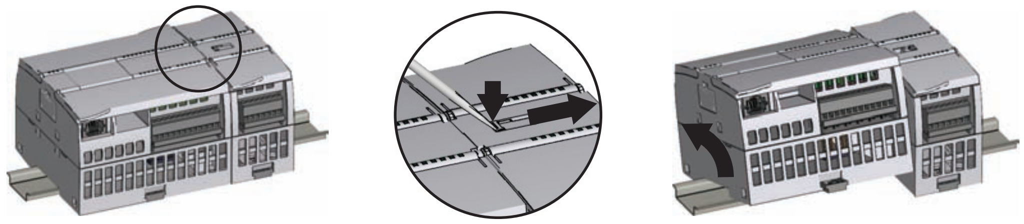
To install the CPU on a DIN rail, follow these steps:



1. Install the DIN rail. Secure the rail to the mounting panel every 75 mm.
2. Hook the CPU over the top of the DIN rail.
3. Pull out the DIN rail clip on the bottom of the CPU to allow the CPU to fit over the rail.
4. Rotate the CPU down into position on the rail.
5. Push in the clips to latch the CPU to the rail.

Removal

To prepare the CPU for removal, remove power from the CPU and disconnect the I/O connectors, wiring, and cables from the CPU. Remove the CPU and any attached communication modules as a unit. All signal modules should remain installed.



If a signal module is connected to the CPU, retract the bus connector:

1. Place a screwdriver beside the tab on the top of the signal module.
2. Press down to disengage the connector from the CPU.
3. Slide the tab fully to the right.

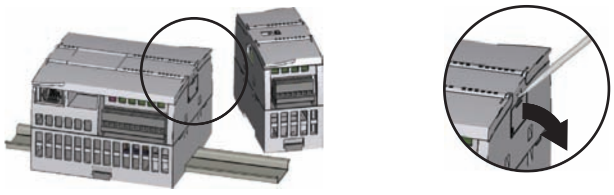
Remove the CPU:

1. Pull out the DIN rail clip to release the CPU from the rail.
2. Rotate the CPU up and off the rail, and remove the CPU from the system.

2.2.2 Installing and removing a signal module

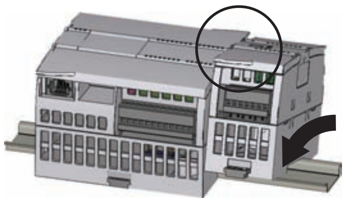
Installation

Install your SM after installing the CPU.



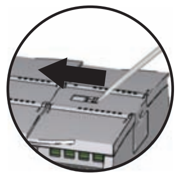
Remove the cover for the connector from the right side of the CPU.

- Insert a screwdriver into the slot above the cover.
- Gently pry the cover out at its top and remove the cover. Retain the cover for reuse.



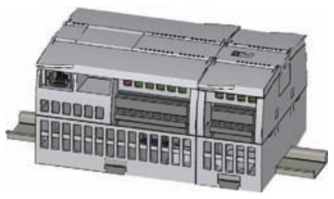
Position the SM beside the CPU.

1. Hook the SM over the top of the DIN rail.
2. Pull out the bottom DIN rail clip to allow the SM to fit over the rail.
3. Rotate the SM down into position beside the CPU and push the bottom clip in to latch the SM onto the rail.



Extend the bus connector.

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.



Extending the bus connector makes both mechanical and electrical connections for the SM.

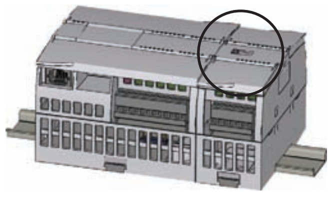
Follow the same procedure to install a signal module to a signal module.

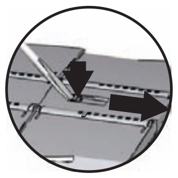
Removal

You can remove any SM without removing the CPU or other SMs in place. To prepare for removing the SM, remove power from the CPU and remove the I/O connectors and wiring from the SM.

Retract the bus connector.

1. Place a screwdriver beside the tab on the top of the SM.
2. Press down to disengage the connector from the CPU.
3. Slide the tab fully to the right.

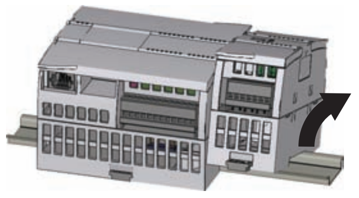




If there is another SM to the right, repeat this procedure for that SM.

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.

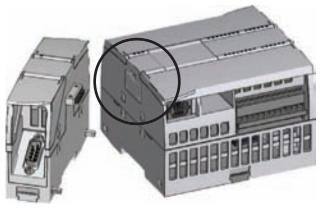
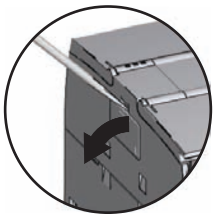
2.2.3 Installing and removing a communication module

Installation

Attach the CM to the CPU before installing the assembly as a unit to the DIN rail or panel.

Remove the bus cover from the left side of the CPU:

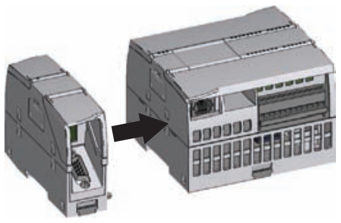
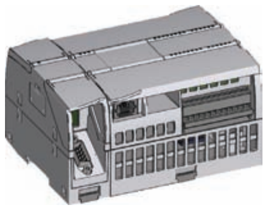
1. Insert a screwdriver into the slot above the bus cover.
2. Gently pry out the cover at its top.

Remove the bus cover. Retain the cover for reuse.

Connect the units:

1. Align the bus connector and the posts of the CM with the holes of the CPU
2. Firmly press the units together until the posts snap into place.

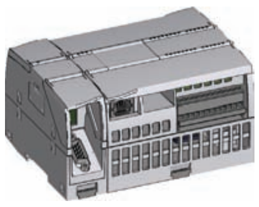
Installing the units on the DIN rail or on a panel.

1. 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.
2. Install the CPU and attached CMs as shown in Installing and removing the CPU (Page 26).
3. 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.

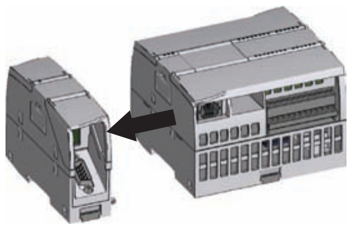
Removal

Remove the CPU and CM as a unit from the DIN rail or panel.



Prepare for CM removal.

1. Remove power from the CPU.
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.



Remove the CM.

1. Grasp the CPU and CMs firmly.
2. Pull them apart.

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

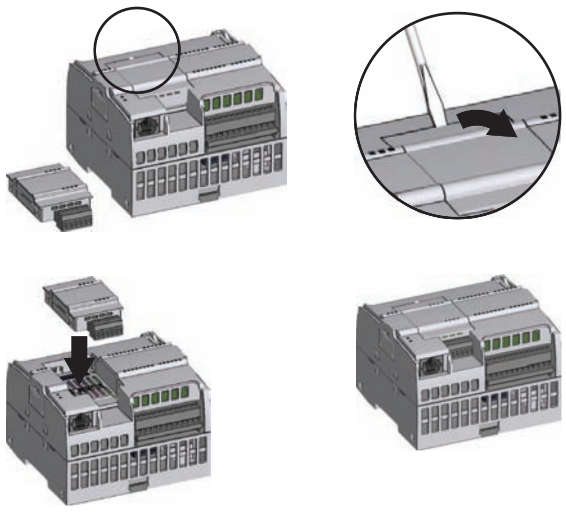
2.2.4 Installing and removing a signal board

Installation

Prepare the CPU for installation of the SB by removing the power from the CPU and removing the top and bottom terminal block covers from the CPU.

To install the SB, follow these steps:

1. Place a screwdriver into the slot on top of the CPU at the rear of the cover.
2. Gently pry the cover up and remove it from the CPU.
3. Place the SB straight down into its mounting position in the top of the CPU.
4. Firmly press the SB into position until it snaps into place.
5. Replace the terminal block covers.

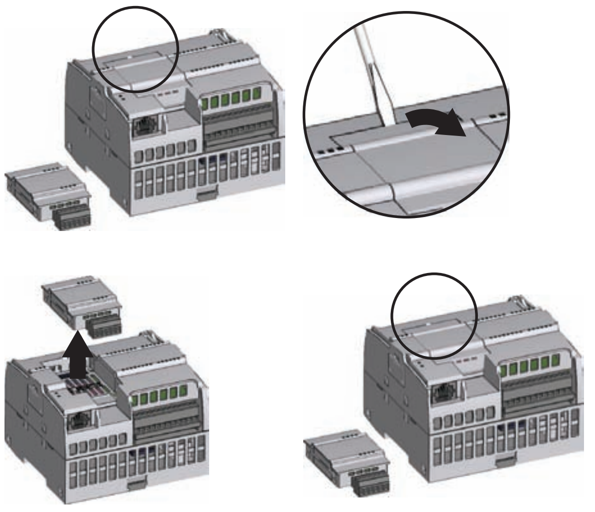


Removal

Prepare the CPU for removal of the SB by removing power from the CPU and removing the top and bottom terminal block covers from the CPU.

To remove the SB, follow these steps:

1. Place a screwdriver into the slot on top of the SB.
2. Gently pry the SB up to disengage it from the CPU.
3. Remove the SB straight up from its mounting position in the top of the CPU.
4. Replace the SB cover.
5. Replace the terminal block covers.



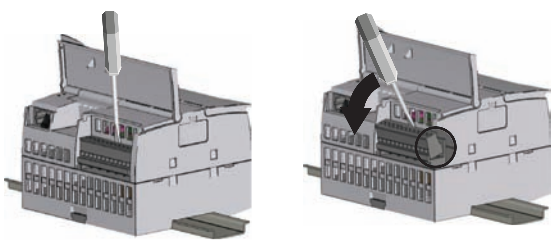
2.2.5 Removing and reinstalling the S7-1200 terminal block connector

The CPU, SB and SM modules provide removable connectors to make connecting the wiring easy. To prepare the system for terminal block connector removal:

- Remove power from the CPU.
- Open the cover above the connector.

To remove the connector, follow these steps:

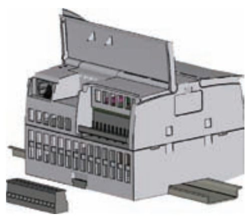
1. Inspect the top of the connector and locate the slot for the tip of the screwdriver.
2. Insert a screwdriver into the slot.
3. Gently pry the top of the connector away from the CPU. The connector will release with a snap.
4. Grasp the connector and remove it from the CPU.

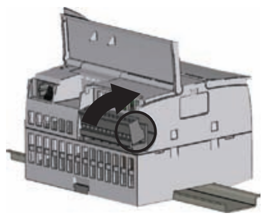


To install the connector, follow these steps:

1. Prepare the components for terminal block installation by removing power from the CPU and opening the cover for the terminal block.
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.





2.3 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 279) 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 connector accepts wire sizes from 2 mm^2 to 0.3 mm^2 (14 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 connector screw is 0.56 N-m (5 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 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.

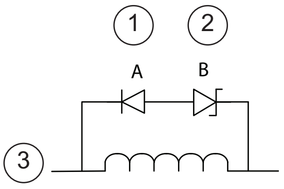
Note

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.

Control DC inductive loads

S7-1200 DC outputs include suppression circuits that are adequate for the inductive loads in most applications. Since the relays can be used for either a DC or an AC load, internal protection is not provided. The following figure shows a sample suppression circuit for a DC load.

In most applications, the addition of a diode (A) across the inductive load is suitable, but if your application requires faster turn-off times, then the addition of a Zener diode (B) is recommended.

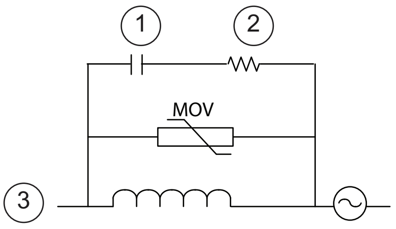


① I1N4001 diode or equivalent
② 8.2 V Zener (DC outputs),
36 V Zener (Relay outputs)
③ Output point

Be sure to size your Zener diode properly for the amount of current in your output circuit.

Relay outputs that control AC loads

When you use a relay output to switch 115 V/230 VAC loads, place resistor/capacitor networks across the AC load as shown in this figure. You can also use a metal oxide varistor (MOV) to limit peak voltage. Ensure that the working voltage of the MOV is at least 20% greater than the nominal line voltage.



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

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.

PLC concepts 3

3.1 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.
- 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.
- Data blocks (DBs) store data that can be used by the program blocks.

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 levels:

- 16 from the program cycle or startup OB
- 4 from time delay interrupt, cyclic 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 limit to the number of blocks supported within the available amount of work memory.

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

The signal board, signal modules and communication modules are detected and logged in only upon power up.

Note

Insertion and extraction of a signal board, signal modules, and communications module under power (hot) is not supported. The only exception is the SIMATIC memory card, which can be inserted or removed while the CPU is under power.

Under the default configuration, all 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.

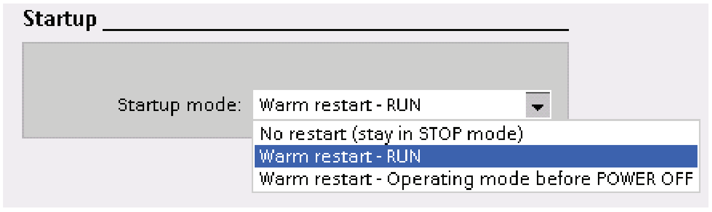
This process provides consistent logic through the execution of the user instructions for a given cycle and prevents the flickering of physical output points that might change state multiple times in the process image output area.

You can specify whether digital and analog I/O points are to be stored in the process image. If you insert a module in the device view, its data is located in the process image of the S7-1200-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.

Configuring the startup parameters

You use the CPU properties to configure how the CPU starts up after a power cycle.


Startup

Startup mode: Warm restart - RUN

No restart (stay in STOP mode)
Warm restart - RUN
Warm restart - Operating mode before POWER OFF

Select whether the CPU starts in STOP mode, RUN mode, or in the previous mode (prior to the power cycle).

The CPU performs a warm restart before going to RUN mode. Warm restart resets all non-retentive memory to the default start values, but retains the current values stored in the retentive memory.

Note

The CPU always performs a restart after a download

Whenever you download an element of your project (such as a program block, data block, or hardware configuration), the CPU performs a restart on the next transition to RUN mode. In addition to clearing the inputs, initializing the outputs and initializing the non-retentive memory, the restart also initializes the retentive memory areas.

After the restart that follows a download, all subsequent STOP-to-RUN transitions perform a warm restart (that does not initialize the retentive memory).

3.1.1 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, and you can download a project.
- In STARTUP mode, the startup OBs (if present) are executed once. Interrupt events are not processed during the startup phase of RUN mode.
- In RUN mode, the scan cycle is executed repeatedly. Interrupt events can occur and be processed at any point within the program cycle phase.

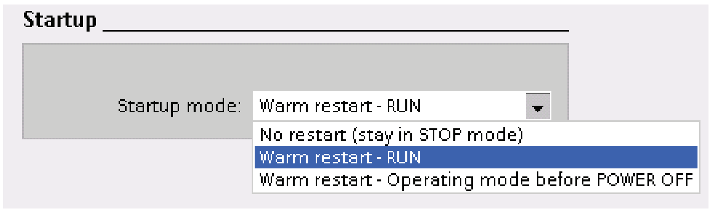
  You cannot download a project while in RUN mode.

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.

You can specify the power-up mode of the CPU complete with restart method using the programming software. 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. The CPU then enters the appropriate power-up mode. Certain detected errors will prevent the CPU from entering the RUN mode. The CPU supports the following power-up modes:

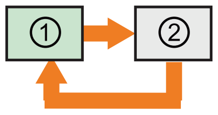
- STOP mode
- Go to RUN mode after warm restart
- Go to previous mode after warm restart


Startup
Startup mode: Warm restart - RUN
No restart (stay in STOP mode)
Warm restart - RUN
Warm restart - Operating mode before POWER OFF

You can change the current operating mode using the "STOP" or "RUN" commands from the online tools of the programming software. You can also include a STP instruction 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.

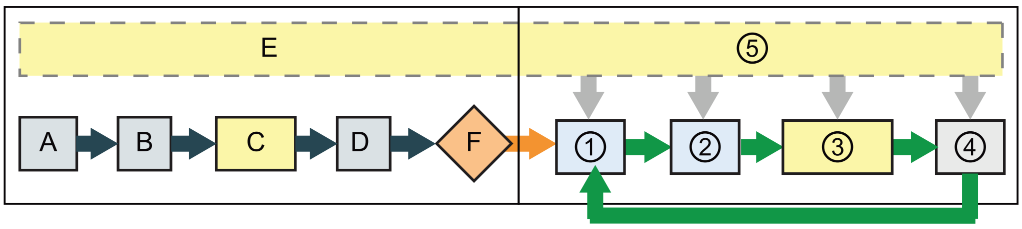
In STOP mode, the CPU ① handles any communication requests (as appropriate) and ② performs self-diagnostics.

In STOP mode, 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 RUN mode, the CPU performs the tasks shown in the following figure.



STARTUP		RUN	
A	Clears the I memory area	①	Writes Q memory to the physical outputs
B	Initializes the outputs with either the last value or the substitute value	②	Copies the state of the physical inputs to I memory
C	Executes the startup OBs	③	Executes the program cycle OBs
D	Copies the state of the physical inputs to I memory	④	Performs self-test diagnostics
E	Stores any interrupt events into the queue to be processed in RUN mode	⑤	Processes interrupts and communications during any part of the scan cycle
F	Enables the writing of Q memory to the physical outputs		

STARTUP processing

Whenever the operating state 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 will 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:

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.

Processing the scan cycle during RUN mode

For each scan cycle, the CPU writes the outputs, reads the inputs, executes the user program, updates communication modules, performs internal housekeeping chores, 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.

Organization blocks (OBs)

OBs control the execution of the user program. Each OB must have a unique OB number. Some 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.
- Time-delay OBs execute at a specified interval after an event is configured by the Start interrupt (SRT_DINT) instruction. The delay time is specified in the input parameter of the extended instruction SRT_DINT. A time-delay OB interrupts normal cyclic program execution when a specified delay time has expired. You can configure up to 4 time-delay events at any given time, with one OB allowed for each configured time-delay event. The time-delay OB 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 4 cyclic interrupt events, with one OB allowed for each configured cyclic interrupt 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.
- Time-error interrupt OBs execute when a time error is detected. A time error interrupt OB will interrupt normal cyclic program execution if the maximum cycle time has been exceeded. The maximum cycle time is defined in the properties of the PLC. OB 80 is the only OB number supported for the time error event. You can configure the action to take when no OB 80 is present: either ignore the error or change to STOP.
- 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. If there is no diagnostic OB in the program, you can configure the CPU to either ignore the error or to change to STOP.

3.1.2 Event execution priorities and queuing

The CPU processing is controlled by events. The events trigger interrupt OBs to be executed. The interrupt OB for an event is specified during the creation of the block, during 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, priority groups 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 OB1) is executed. The other program cycle OBs are executed sequentially, in numerical order, within the program cycle.

The cyclic interrupt events allows you to configure the execution of an interrupt OB at a configured time interval. The time interval is configured when the OB is created and selected to be a cyclic interrupt OB. The cyclic events will interrupt the program cycle and execute the cyclic interrupt OB (the cyclic event is in a higher priority group than the program cycle event). Only one cyclic interrupt OB can be attached to a cyclic event. The CPU supports four cyclic interrupt events. The cyclic interrupt OBs have a phase shift property so that the execution of cyclic interrupts with the same time period can be offset from one another by the phase shift amount.

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 a higher priority group 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, and the event priorities are classified into priority groups. The following table summarizes the queue depths, priority groups and priorities for the supported CPU events.

Note

You cannot change the priority or the priority group assignments or the queue depths.

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.

Event Type (OB)	Quantity	Valid OB Numbers	Queue Depth	Priority Group	Priority
Program Cycle	1 program cycle event Multiple OBs allowed	1 (default) 200 or greater	1	1	1
Startup	1 startup event [1] Multiple OBs allowed	100 (default) 200 or greater	1		1
Time Delay	4 time delay events 1 OB per event	200 or greater	8	2	3
Cyclic	4 cyclic events 1 OB per event	200 or greater	8		4
Edges	16 rising edge events 16 falling edge events 1 OB per event	200 or greater	32		5
HSC	6 CV = PV events 6 direction changed events 6 external reset events 1 OB per event	200 or greater	16		6
Diagnostic Error	1 event	82 only	8		9
Time Error event/ MaxCycle time event	1 time error event 1 MaxCycle time event	80 only	8	3	26
2xMaxCycle time event	1 2xMaxCycle time event	No OB called	-	3	27

[1] Special cases for the startup event

- 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).
- No events are allowed to interrupt the startup event. Events that occur during the startup event are instead queued for later processing after the startup event is finished.

After the execution of an OB has started, processing of the OB cannot be interrupted by the occurrence of another event from the same or lower priority group. Such events are queued for later processing, allowing the current OB to finish.

However, an event from a higher priority group will interrupt the current OB, and the CPU then executes the OB for the higher-priority event. After the higher-priority OB finishes, the CPU executes the OBs for any other events queued in this higher priority group, based on the priority within that group. When no other events are pending (queued) in this higher priority group, the CPU then returns to the lower priority group and resumes the processing of the pre-empted OB at the point where the processing of that OB had been interrupted.

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 210 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 43)" 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 or a time-delay 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 OB 80 does not exist, then the CPU ignores the error. 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 43)

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. The following startup locations are supported by 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

Some 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

All 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".

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. The following startup locations are supported by OB 82:

Input	Data type	Description
IOstate	WORD	IO state of the device
laddr	HW_ANY	Hardware identifier of the device or functional unit that reported the error
channel	UINT	Channel number
multierror	BOOL	TRUE if more than one error is present (*not supported in early releases*)

Bit 4 of the IO_state indicates whether the event is due to the occurrence or removal of an error. Bit 4 is 1 if an error is present (example: wire break) and is 0 if the error is no longer present.

The ladder input contains the hardware identifier (HW ID) of the device or functional unit which returned the error. The HW ID 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 HW ID. These entries in the Constants tab of the PLC tags cannot be changed.

The channel number begins at 0 for the first input point (analog or digital) and begins at 64 for the first output point (analog or digital). The different offsets are necessary to distinguish inputs from outputs in the event the device contains both. If an error affects the complete device or functional unit, such as no user power, the most-significant bit of the channel-number word is set (channel number 32768).

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 no time error interrupt OB 80 is present, then the CPU generates an error and continues to execute the user program
- If a time error interrupt OB 80 is present, 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.

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

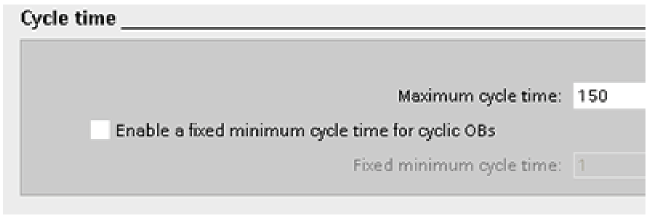
[1] The maximum scan cycle time is always enabled. Configure a cycle time between 1 ms to 6000 ms. The default is 150 ms.

[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.



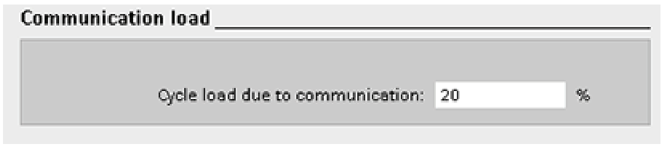
Cycle time

Maximum cycle time: 150

Enable a fixed minimum cycle time for cyclic OBs

Fixed minimum cycle time: 1

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



Communication load

Cycle load due to communication: 20 %

For more information about the scan cycle, see "Monitoring the cycle time". (Page 43)

3.1.3 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 occurs, the CPU has enough hold-up time to retain the values of a limited number of specified locations. These retentive values are then restored 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 the "Symbolic access only" box selected, then the interface editor for this FB includes a "Retain" column. In this column, you can select either "Retain" or "Non-Retain" 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, but only for display; you cannot change the retentive state from within the instance DB interface editor for an FB that was configured to be "Symbolic access only".

  If an FB was created with the "Symbolic access only" box deselected, 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 **not** be "Symbolic access only", 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 "symbolic access only". This option can only be selected when the FB is created. To determine whether an existing FB was configured for "symbolic access only", right-click the FB in the Project tree, select "Properties", and then select "Attributes".
- 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 setting for symbolic addressing you can define the retentive state either for individual or for all tags of a global data block.
  – If the "Symbolic access only" attribute of the DB is checked, the retentive state can be set for each individual tag.
  – If the "Symbolic access only" attribute of the DB is un-checked, 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.

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.

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 2 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 Basic 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.

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.

- 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:
  - "Always 0 (low)" bit is always set to 0.
  - "Always 1 (high)" bit is always set to 1.
  - "Diagnostic graph changed" 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 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.
  - "First scan" 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.)
- 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.

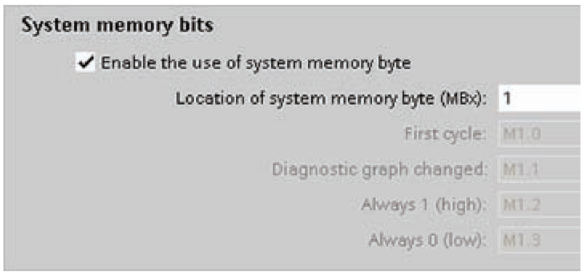
The CPU initializes these bytes on the transition from STOP mode to STARTUP mode. The bits of the clock memory change synchronously to the CPU clock throughout the STARTUP and RUN modes.

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.



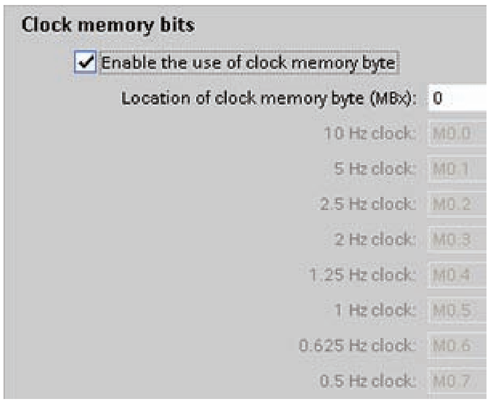
System memory bits

✓ Enable the use of system memory byte

Location of system memory byte (MBx):	1
First cycle:	M1.0
Diagnostic graph changed:	M1.1
Always 1 (high):	M1.2
Always 0 (low):	M1.3

System memory configures a byte that turns on (value = 1) for the following conditions.

- First scan: Turns on for the first scan cycle in run mode
- Diagnostic graph changed:
- Always 1 (high): Always turned on
- Always 0 (low): Always turned off



Clock memory bits

✓ Enable the use of clock memory byte

Location of clock memory byte (MBx):	0
10 Hz clock:	M0.0
5 Hz clock:	M0.1
2.5 Hz clock:	M0.2
2 Hz clock:	M0.3
1.25 Hz clock:	M0.4
1 Hz clock:	M0.5
0.625 Hz clock:	M0.6
0.5 Hz clock:	M0.7

Clock memory configures a byte that cycles the individual bits on and off at fixed intervals.

The clock flags each generate 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.

Configuring the behavior of output values when the CPU is in STOP mode

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.

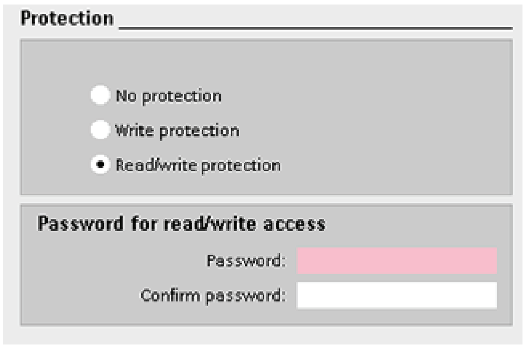
- Freezing the outputs to remain in last state: The outputs retain their current value at the time of the transition from RUN to STOP. After power up, the outputs are set to the default substitute value.

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.

3.1.4 Password protection for the S7-1200 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.



Protection

No protection
Write protection
[✓ Read/write protection]

Password for read/write access

Password:
Confirm password:

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.

The password is case sensitive.

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.

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).

3.1.5 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 Basic to the CPU.

For information about the creation and use of an empty transfer card, see the section of transfer cards (Page 63).

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.

3.2 Data storage, memory areas and addressing

The CPU provides several options for storing data during the execution of the user program:

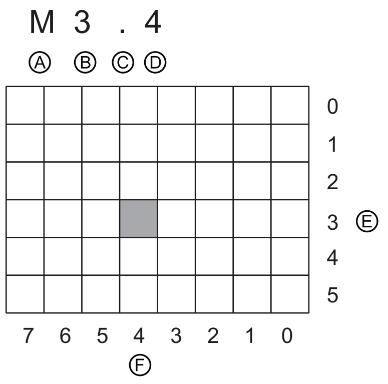
- Global memory: The CPU provides a variety of specialized memory areas, including inputs (I), outputs (Q) and bit memory (M). This memory is accessible by all code blocks without restriction
- Data block (DB): You can include DBs in your user program to store data for the code blocks. The data stored 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.

Memory area	Description	Force	Retentive
I Process image input	Copied from physical inputs at the beginning of the scan cycle	No	No
I_:P (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 (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

Memory area	Description	Force	Retentive
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

Each different memory location has a unique address. Your user program uses these addresses to access the information in the memory location. The figure shows how to access a bit (which is also called "byte.bit" addressing). In this example, the memory area and byte address (I = input, and 3 = byte 3) are followed by a period (".") to separate the bit address (bit 4).



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

You can access data in most memory areas (I, Q, M, DB, and L) as bytes, words, or double words by using the "byte address" format. To access a byte, word, or double word of data in the memory, you must specify the address in a way similar to specifying the address for a bit. This includes an area identifier, data size designation, and the starting byte address of the byte, word, or double word value. Size designators are B (byte), W (word) and D (double-word), for example, IB0, MW20, or QD8. References such as I0.3 and Q1.7 access the process image. To access the physical input or output, append the reference with ":P" (such as I0.3:P, Q1.7:P, or "Stop:P").

Accessing the data in the memory areas of the CPU

STEP 7 Basic 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, an initial 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.

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.

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.

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.

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.

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:

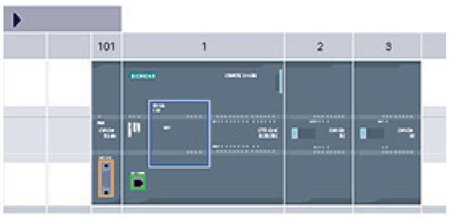
- 16 Kbytes for startup and program cycle, including associated FBs and FCs
- 4 Kbytes for standard interrupt events including FBs and FCs
- 4 Kbytes for error interrupt events including FBs and FCs

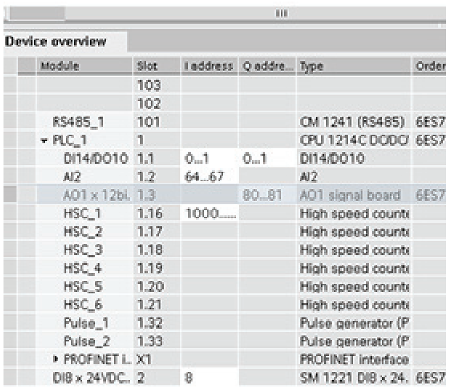
You access temp memory by symbolic addressing only.

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 specify a data block to be either read/write or read only. 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.

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

Addressing 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 count	
HSC_2	1.17			High speed count	
HSC_3	1.18			High speed count	
HSC_4	1.19			High speed count	
HSC_5	1.20			High speed count	
HSC_6	1.21			High speed count	
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 complete 8 bit bytes, whether the module uses all the points or not. Analog inputs and outputs are assigned in groups of 2 points (4 bytes). In this example, you could change the address of the DI16 to 2..3 instead of 8..9. The tool will assist you by changing address ranges that are the wrong size or conflict with other addresses.

The figure shows an example of a CPU 1214C with two SMs.

3.3 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 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, or 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.

You can also enter a constant value for many of the input parameters. The following table describes the supported elementary data types including examples of constant entry. All except String are available in the PLC tags editor and the block Interface editors. String is available only in the block Interface editors. The following table defines the elementary data types.

Data type	Size (bits)	Range	Constant Entry Examples
Bool	1	0 to 1	TRUE, FALSE, 0, 1
Byte	8	16#00 to 16#FF	16#12, 16#AB
Word	16	16#0000 to 16#FFFF	16#ABCD, 16#0001
DWord	32	16#00000000 to 16#FFFFFFFF	16#02468ACE
Char	8	16#00 to 16#FF	'A', 't', '@'
Sint	8	-128 to 127	123, -123
Int	16	-32,768 to 32,767	123, -123
Dint	32	-2,147,483,648 to 2,147,483,647	123, -123
USInt	8	0 to 255	123
UInt	16	0 to 65,535	123
UDInt	32	0 to 4,294,967,295	123
Real	32	+/-1.18 x 10 $^{-38}$ to +/-3.40 x 10 ^{38}	123.456, -3.4, -1.2E+12, 3.4E-3
LReal	64	+/-2.23 x 10^{-308} to +/-1.79 x 10^{308}	12345.123456789 -1.2E+40
Time	32	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 5#-2d T#1d_2h_15m_30x_45ms
String	Variable	0 to 254 byte-size characters	'ABC'

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

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

Format for real numbers

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.

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.

Format for the string data type

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 when declaring your string; for example, "MyString[10]" would specify a 10-byte maximum size for MyString. If you do not include the brackets with a max size specifier, 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.

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

Arrays

You can create an array that contains multiple elements of an elementary 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, 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 elementary data types, such as BOOL, SINT, UDINT

Negative indices are supported. You can name the array in the Name column of the block interface editor. The following table shows examples of arrays as they might appear in the block interface editor:

Name	Data type	Comment
My_Bits	Array [1 .. 10] of BOOL	This array contains 10 Boolean values
My_Data	Array [-5 .. 5] of SINT	This array contains 11 SINT values, including index 0

You reference elements of arrays in your program using the following syntax:

- Array_name[*i*], where *i* is the desired index.

Examples as they might appear in the program editor as a parameter input:

- #My_Bits[3] - references the third bit of array "My_Bits"
- #My_Data[-2] - references the fourth SINT of array "My_Data"

The # symbol is inserted automatically by the program editor.

DTL (Data and Time Long) data type

The DTL data type is a structure of 12 bytes that saves information on date and time in a predefined structure. You can define a DTL in either the Temp memory of the block or in a DB.

Length (bytes)	Format	Value range	Example of value input
12	Clock and calendar (Year-Month tag: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.

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	Day of week	USINT	1(Sunday) to 7(Saturday) The weekday is not considered in the value entry.
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			

3.4 Using a memory card

NOTICE
The CPU supports only the pre-formatted SIMATIC memory card (Page 318). If you use a Windows formatter to reformat the SIMATIC memory card, the CPU cannot use the reformatted memory card. 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 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 Basic. 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 53). 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 Basic applies the force values only to the external load memory on the program card.

3.4.1 Inserting a memory card in the CPU

WARNING

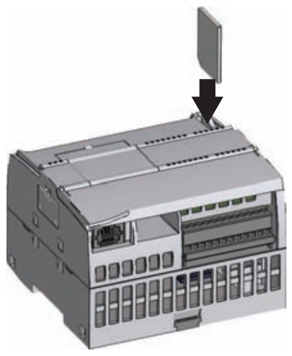
If you insert a memory card (whether configured as a program or transfer card) into a running CPU, the CPU goes immediately to STOP mode. Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment. Always install an emergency stop circuit for your application or process.

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.

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.



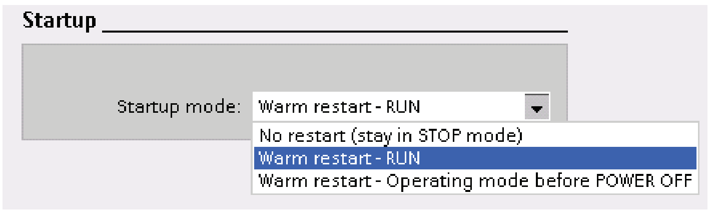
Check that the memory card is not write-protected. Slide the protection switch away from the "Lock" position.

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. Please disregard this message. The evaluation of the memory card does not start until you change the CPU to RUN mode, reset the CPU memory with an MRES, or power-cycle the CPU.

3.4.2 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).


Startup
Startup mode:
No restart (stay in STOP mode)
Warm restart - RUN
Warm restart - Operating mode before POWER OFF

Startup mode: Warm restart - RUN

3.4.3 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

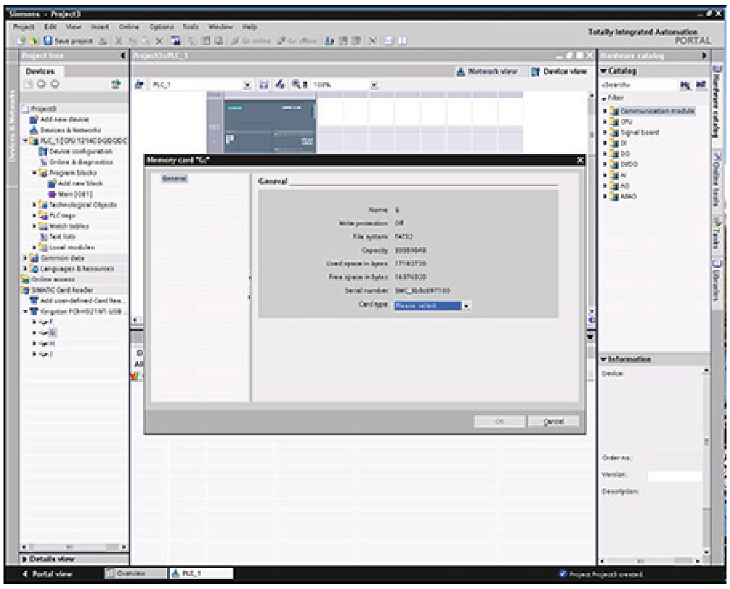
Always remember to configure the startup parameter of the CPU (Page 63) before copying a program to the transfer card. To create a transfer card, follow these steps:

1. Insert a blank memory card into the card reader/writer attached to your programming device.

   (If the memory card is not blank, delete the "SIMATIC.S7S" folder and the "S7_JOB.S7S" file on the memory card using an application such as Windows Explorer.)

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 memory card in the card reader and selecting "Properties" from the context menu.
4. In the "Memory card" dialog, select "Transfer" from the drop-down menu.

   At this point, STEP 7 Basic creates the empty transfer card. If you are creating an empty transfer card, such as to recover from a lost CPU password (Page 53), remove the transfer card from the card reader.



General

Name: G
Write protection: Off
File system: FAT32
Capacity: 30589048
Used space in bytes: 17182720
Free space in bytes: 14376820
Serial number: SMC_[illegible]
Card type: Please select...

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

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

1. Insert the transfer card into the CPU (Page 62). If the CPU is in RUN, the CPU will go to STOP mode. (The maintenance LED flashes to indicate that the memory card needs to be evaluated.)
2. Use one of the following options to evaluate the memory card:
   – Power-cycle the CPU.
   – Perform a STOP-to-RUN transition.
   – Perform a memory reset (MRES).
3. After the rebooting and evaluating the memory card, the CPU copies the program to the internal load memory of the CPU. When the copy operation is complete, the CPU flashes the maintenance LED to indicate that the transfer card can be removed.
4. Remove the "Transfer" card from the CPU.
5. Use one of the following options to evaluate the new program transferred to internal load memory:
   – Power-cycle the CPU.
   – Perform a STOP-to-RUN transition.
   – Perform a memory reset (MRES).

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.

3.4.4 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.

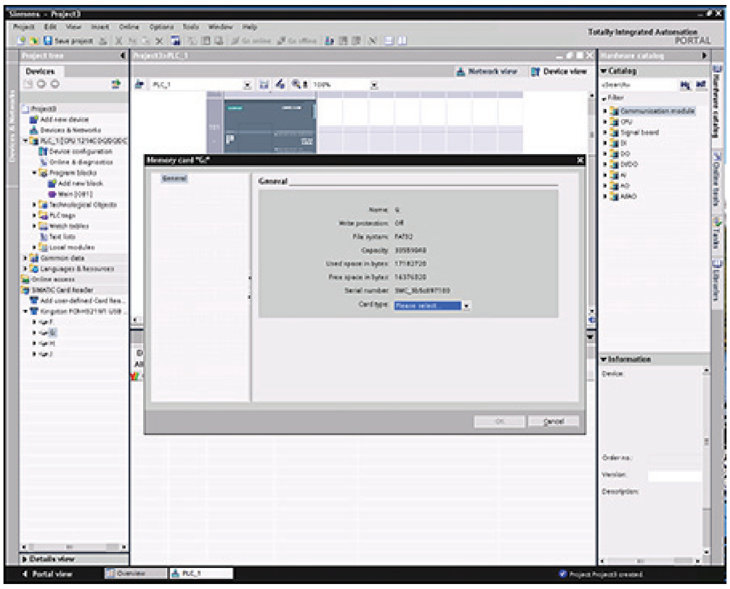
Note

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 63) before copying a project to the program card. To create a program card with STEP 7 Basic, follow these steps:

1. Insert a blank memory card into the card reader/writer attached to your programming device.

   (If the memory card is not blank, delete the "SIMATIC.S7S" folder and the "S7_JOB.S7S" file on the memory card using an application such as Windows Explorer.)
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 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.



Memory card "G:"

General

Name: G
Write protection: Off
File system: FAT32
Capacity: 31559048
Used space in bytes: 17182720
Free space in bytes: 14376328
Serial number: SMC_[illegible]
Card type: Please select...

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

CAUTION
If you insert a blank memory card into the CPU, the CPU goes to STOP mode. If you power-cycle the CPU, change the CPU from STOP to RUN mode, or reset the CPU memory (MRES), the CPU copies the internal load memory of the CPU to the memory card (which configure the memory card as a program card) and erases the program from the internal load memory. If you remove the program card, the CPU has no program in the internal load memory.

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 LED flashes to indicate that the program card needs to be evaluated
2. Use one of the following options to evaluate the program card:
   – Power-cycle the CPU.
   – Perform a STOP-to-RUN transition.
   – Perform a memory reset (MRES).
3. The CPU reboots itself. After rebooting and evaluating the program card, the CPU erases the internal load memory of the CPU.

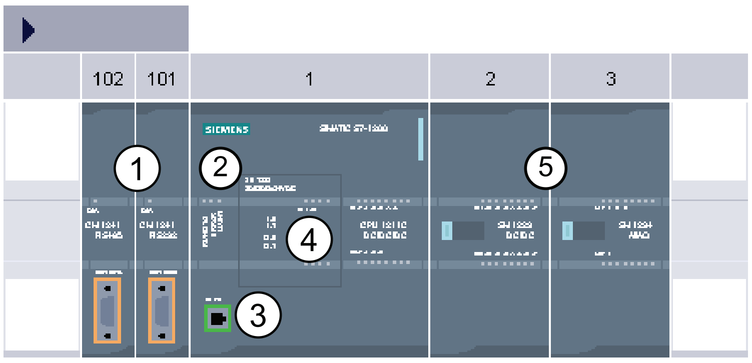
The CPU then goes to the start-up mode (RUN or STOP) that you configured for the CPU.

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 Device configuration

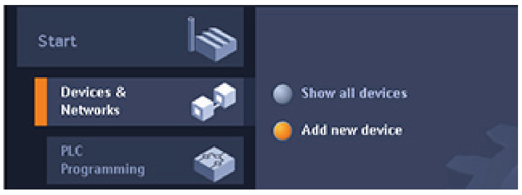
You create the device configuration for your PLC by adding a CPU and additional modules to your project.



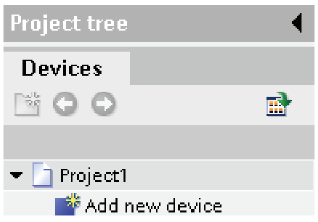
①	Communications module (CM): Up to 3, inserted in slots 101, 102, and 103
②	CPU: Slot 1
③	Ethernet port of CPU
④	Signal board (SB): up to 1, inserted in the CPU
⑤	Signal module (SM) for digital or analog I/O: up to 8, inserted in slots 2 through 9 (CPU 1214C allows 8, CPU 1212C allows 2, CPU 1211C does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

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

- In the Portal view, select "Devices & Networks" and click "Add device".



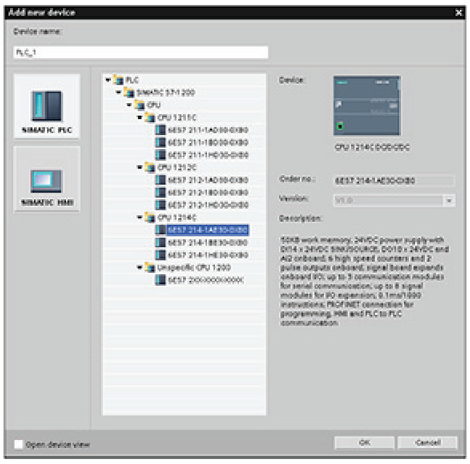
- In the Project view, under the project name, double-click "Add new device".



4.2 Inserting a CPU

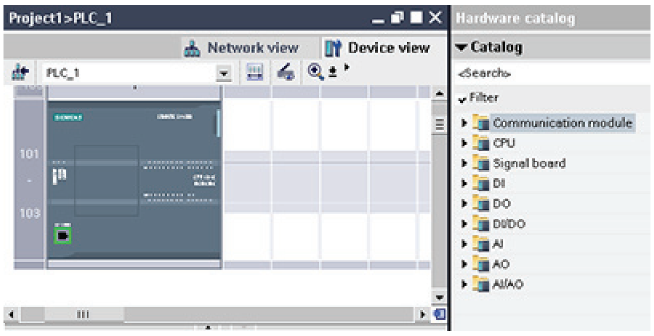
You create your device configuration by inserting a CPU into your project. Selecting the CPU from the "Add a new device" dialog creates the rack and CPU.

"Add a new device" dialog

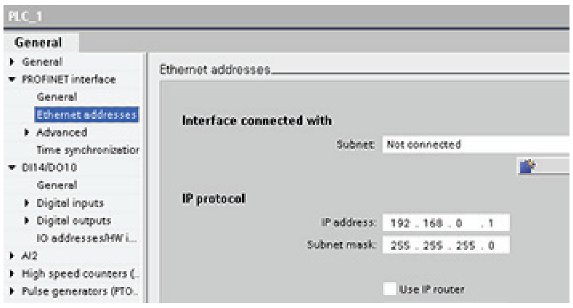


Device name: PLC_1
Device:
Order no.: 6ES7 214-1AE30-0XB0
Version: V1.0
Description:
50KB work memory; 24VDC power supply with DI14 x 24VDC SINK/SOURCE, DO10 x 24VDC and AI2 onboard; 6 high speed counters and 2 pulse outputs onboard; signal board expands onboard I/O; up to 3 communication modules for serial communication; up to 8 signal modules for I/O expansion; 0.1ms/1000 instructions; PROFINET connection for programming, HMI and PLC to PLC communication

Device view of the hardware configuration



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



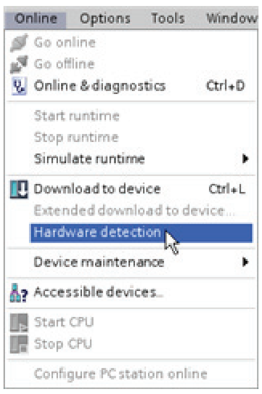
Subnet: Not connected
IP address: 192 . 168 . 0 . 1
Subnet mask: 255 . 255 . 255 . 0

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.

4.3 Detecting the configuration for an unspecified CPU

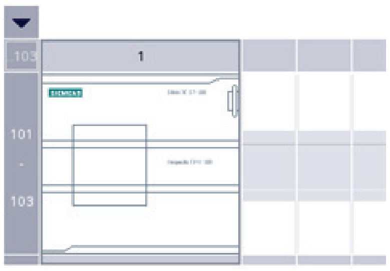
Uploading an existing hardware configuration is easy

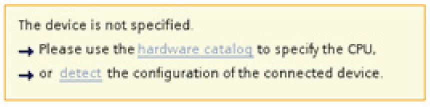


If you are connected to a CPU, you can upload the configuration of that CPU, including any modules, to your project. Simply create a new project and select the "unspecified CPU" instead of selecting a specific CPU. (You can also skip the device configuration entirely by selecting the "Create a PLC program"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

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

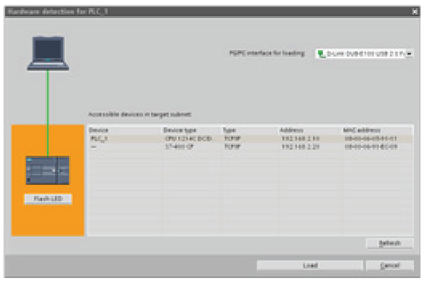
From the device configuration editor, you select the option for detecting the configuration of the connected device.

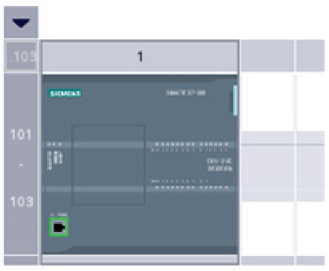




The device is not specified.
→ Please use the hardware catalog to specify the CPU,
→ or detect the configuration of the connected device.

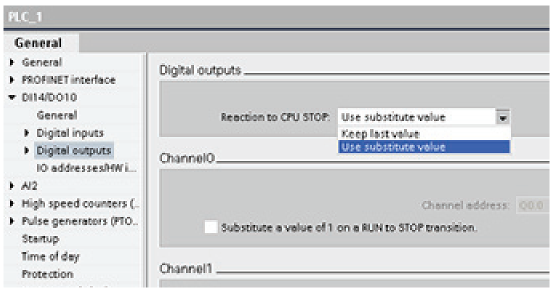
After you select the CPU from the online dialog, STEP 7 Basic uploads the hardware configuration from the CPU, including any modules (SM, SB, or CM). You can then configure the parameters for the CPU and the modules.





4.4 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.



Reaction to CPU STOP: Use substitute value

Channel address: Q0.0

☐ Substitute a value of 1 on a RUN to STOP transition.

Edit the properties to configure the following parameters:

- 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 and pulse generators: Enables and configures the high-speed counters (HSC) and the pulse generators used for pulse-train operations (PTO) and pulse-width modulation (PWM)

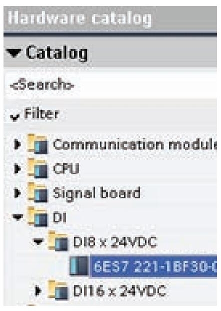
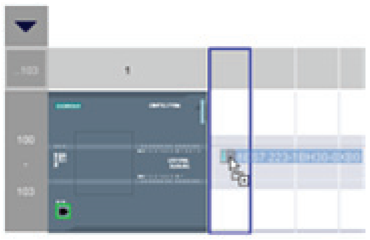
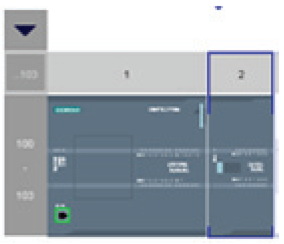
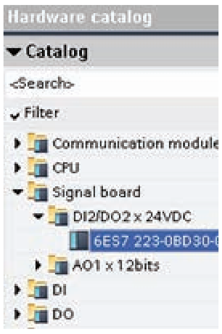
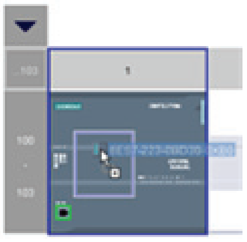
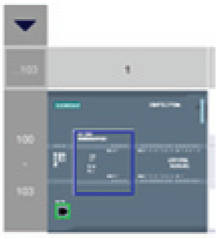
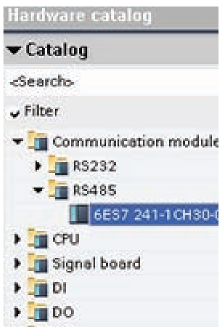
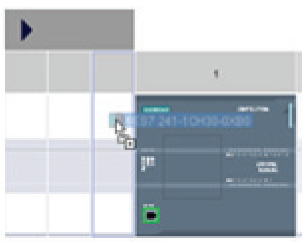
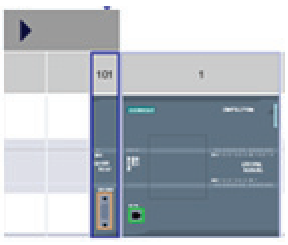
  When you configure the outputs of the CPU or signal board as pulse generators (for use with the PWM or basic motion control instructions), the corresponding outputs addresses (Q0.0, Q0.1, Q4.0, and Q4.1) are removed from the Q memory and cannot be used for other purposes in your user program. If your user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.
- Startup: 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
- Time of day: Sets the time, time zone and daylight saving time
- Protection: Sets the read/write protection and password for accessing the CPU
- System and clock memory: Enables a byte for "system memory" functions (for a "first-scan" bit, an "always-on" bit, and an "always-off" bit) and enables a byte for "clock memory" functions (where each bit toggles on and off at a predefined frequency).
- Cycle time: Defines a maximum cycle time or a fixed minimum cycle time
- Communications load: Allocates a percentage of the CPU time to be dedicated to communication tasks

4.5 Adding modules to the configuration

Use the hardware catalog to add modules to the CPU. There are three types of modules:

- Signal modules (SM) provide additional digital or analog I/O points. These modules are connected to the right side of the CPU.
- Signal boards (SB) provide just a few additional I/O points for the CPU. The SB is installed on the front of the CPU.
- Communication modules (CM) provide an additional communication port (RS232 or RS485) for 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.

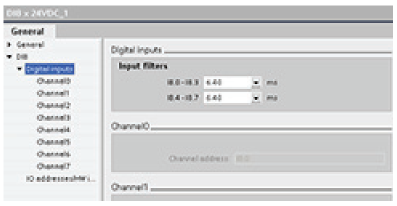
Module	Select the module	Insert the module	Result
SM	 Hardware catalog Catalog <Search> Filter Communication modul CPU Signal board DI DI8 x 24VDC 6ES7 221-1BF30- DI16 x 24VDC		
SB	 Hardware catalog Catalog <Search> Filter Communication module CPU Signal board DI2/DO2 x 24VDC 6ES7 223-0BD30- AO1 x 12bits DI DO		
CM	 Hardware catalog Catalog <Search> Filter Communication module RS232 RS485 6ES7 241-1CH30- CPU Signal board DI DO		

4.6 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.

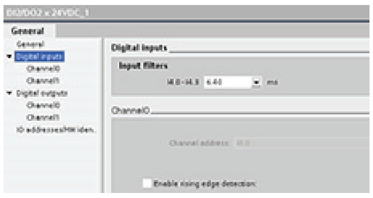
Configuring a signal module (SM) or a signal board (SB)

- 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.



I0.0 - I0.3: 6.40 ms
I0.4 - I0.7: 6.40 ms
Channel address: I0.0

- 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. 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

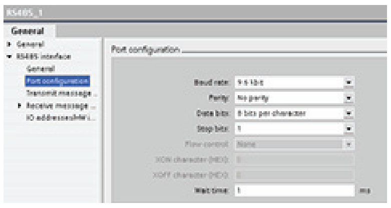


I4.0 - I4.3: 6.40 ms
Channel address: I4.0
☐ Enable rising edge detection:

- I/O diagnostic addresses: Configures the start address for the set of inputs and outputs of the module

Configuring a communication module (CM)

- Port configuration: Configures the communication parameters, such as baud rate, parity, data bits, stop bits, flow control, XON and XOFF characters, and wait time
- Transmit message configuration: Enables and configures transmit related options
- Receive message configuration: Enables and configures the message-start and message-end parameters

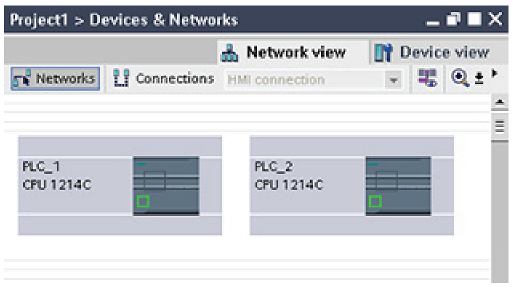
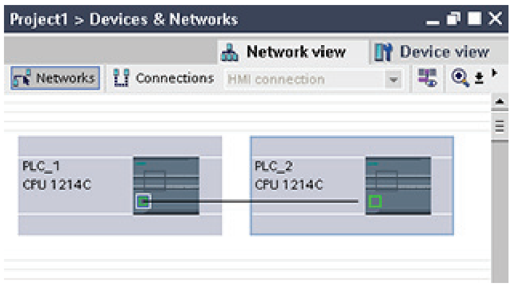
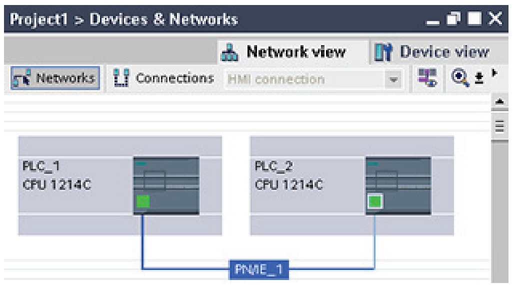


Baud rate: 9.6 kbit/s
Parity: No parity
Data bits: 8 bits per character
Stop bits: 1
Flow control: None
XON character (HEX): 0
XOFF character (HEX): 0
Wait time: 1 ms

These configuration parameters can be changed by your program.

4.7 Creating a network connection

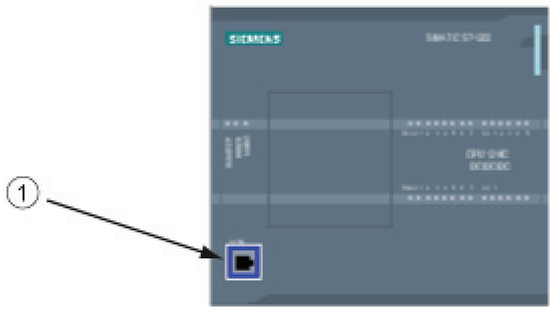
Use the "Network view" of Device configuration to create the network connections between the devices in your project. After creating the network connection, use the "Properties" tab of the inspector window to configure the parameters of the network.

Action	Result
Select "Network view" to display the devices to be connected.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C PLC_2 CPU 1214C
Select the port on one device and drag the connection to the port on the second device.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C PLC_2 CPU 1214C
Release the mouse button to create the network connection.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C PLC_2 CPU 1214C PN/IE_1

4.8 Configuring an IP address in your project

Configuring the PROFINET interface

After you configure the rack with the CPU (Page 72) , you can configure parameters for the PROFINET interface. To do so, click the green PROFINET box on the CPU to select the PROFINET port. 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.

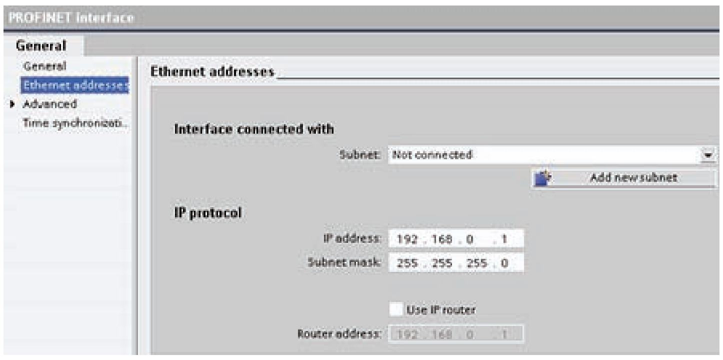
Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet tend to be located in close physical proximity to each other on a Local Area Network (LAN). A mask (known as the subnet mask or network mask) defines the boundaries of an IP subnet.

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.



Ethernet addresses

Interface connected with

Subnet: Not connected

IP protocol

IP address: 192 . 168 . 0 . 1

Subnet mask: 255 . 255 . 255 . 0

Use IP router

Router address: 192 . 168 . 0 . 1

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

Note

The CPU does not have a pre-configured IP address. You must manually assign an IP address for the CPU. If your CPU is connected to a router on a network, you must also enter the router's IP address. All IP addresses are configured when you download the project.

Refer to "Assigning IP addresses to programming and network devices (Page 212)" for more information.

The following table defines the 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 Programming concepts

5.1 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.

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.

5.2 Structuring your user program

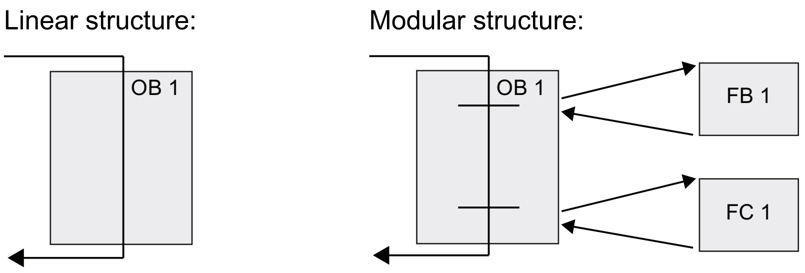
When you create a user program for the automation tasks, you insert the instructions for the program into code blocks:

- 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 of 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.



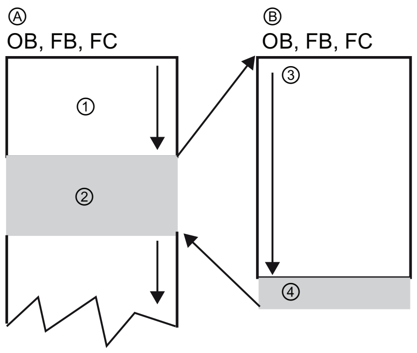
By creating generic code blocks that can be reused within the user program, you can simplify the design and implementation of the user program. Using generic code blocks has a number of benefits:

- 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.

5.3 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.

A	Calling block
B	Called (or interrupting) block
①	Program execution
②	Operation that calls another block
③	Program execution
④	Block end (returns to calling block)



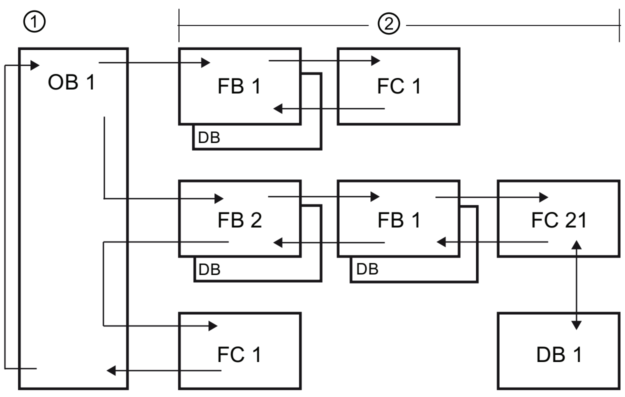
When a code block calls another code block, the CPU executes the program code in the called block. After execution of the called block is complete, the CPU resumes the execution of the calling block.

Processing continues with execution of the instruction that follows after the block call.

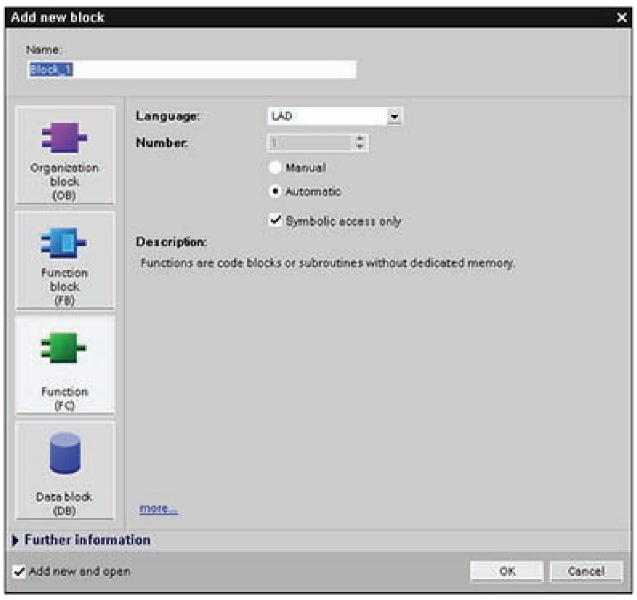
You can nest the block calls for a more modular structure.

① Start of cycle

② Nesting depth



Creating reusable code blocks



Name: Block_1

Language: LAD

Number: 1

( ) Manual

(●) Automatic

✓ Symbolic access only

Description:

Functions are code blocks or subroutines without dedicated memory.

✓ Add new and open

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

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

5.3.1 Organization block (OB)

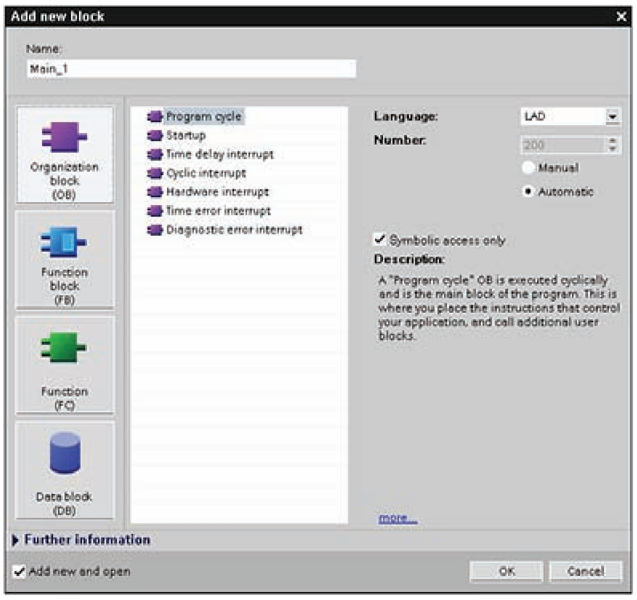
Organization blocks provide structure for your program. They serve as the interface between the operating system and the user program. OBs are event driven. An event, such as a diagnostic interrupt or a time interval, will cause the CPU to execute an OB. Some OBs have predefined start events and behavior.

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 startup tasks, 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.



Name: Main_1

Language: LAD

Number: 200

Manual [✓ Automatic]

✓ Symbolic access only

Description:

A "Program cycle" OB is executed cyclically and is the main block of the program. This is where you place the instructions that control your application, and call additional user blocks.

✓ Add new and open

Depending on their respective priority levels, one OB can interrupt another OB. 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.

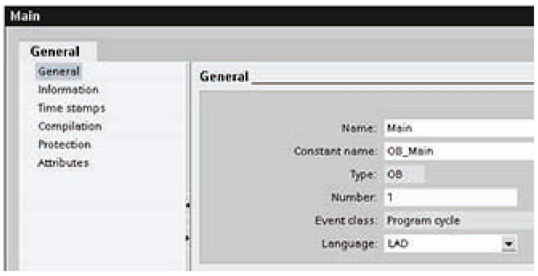
The CPU determines the order for handling interrupt events by a priority assigned to each OB. Each event has a particular servicing priority. Several interrupt events can be combined into priority classes. For more information, refer to the PLC concepts chapter section on execution of the user program (Page 37).

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 a OB. Enter the name for your OB and enter an OB number greater than 200.

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

Configuring the operation of an OB



Name: Main
Constant name: OB_Main
Type: OB
Number: 1
Event class: Program cycle
Language: LAD

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.

5.3.2 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.

Use FCs to perform the following tasks:

- To perform standard and reusable operations, such as for mathematical calculations.
- To perform 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.

5.3.3 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 (IN), output (OUT), and in/out (IN_OUT) parameters in an instance DB.

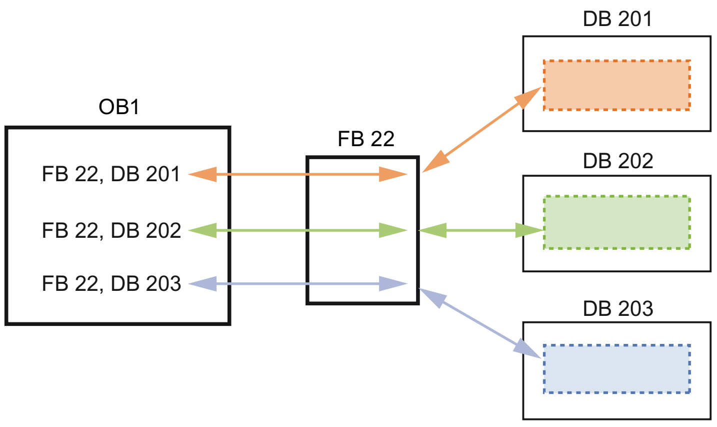
Assigning initial values

If the input, output, or in/out parameters of a function block (FB) are not assigned with values, the values stored in the instance data block (DB) will be used. In some cases, you must assign parameters.

You can assign initial values to the parameters in the FB interface. These values are transferred to the associated instance DB. If you do not assign parameters, the values currently stored in the instance DB will be used.

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.



5.3.4 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). You can define a DB as being read-only.

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.

5.4 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 and EN_AIRT 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 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 issues. 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, and the PROFINET instructions (such as TSEND_C and TRCV_C) 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

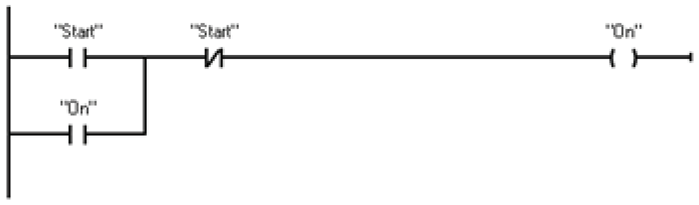
The use of the DIS_AIRT instruction delays the processing of interrupt OBs until the EN_AIRT instruction is executed, affecting the interrupt latency (time from an event to the time when the interrupt OB is executed) of your user program.

5.5 Selecting the programming language

You have the option of choosing either the LAD (ladder logic) or FBD (Function Block Diagram) programming language.

LAD programming language

LAD is a graphical programming language. The representation is based on circuit diagrams.



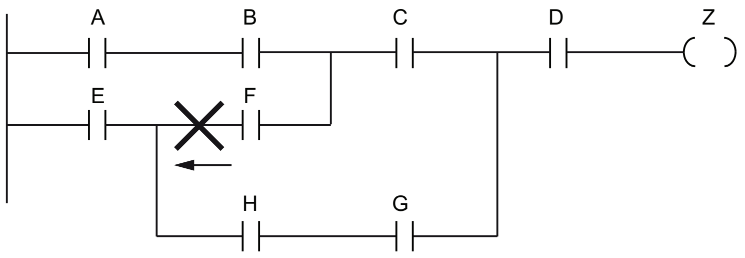
The elements of a circuit diagram, such as normally closed and normally open contacts, and coils are linked to form networks.

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.

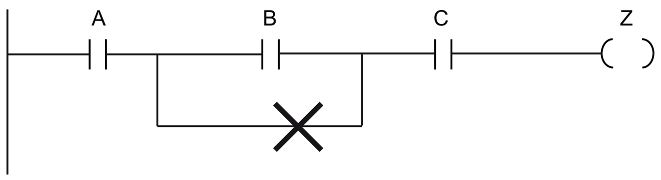
LAD provides "box" instructions for a variety of functions, such as math, timer, counter, and move.

Consider the following rules when creating a LAD network:

- Every LAD network must terminate with a coil or a box instruction. Do not terminate a network with either a Compare instruction or an Edge-detection (Positive-edge or Negative-edge) instruction.
- You cannot create a branch that could result in a power flow in the reverse direction.



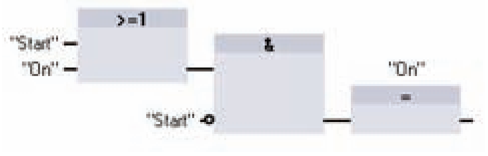
- You cannot create a branch that would cause a short circuit.



Function Block Diagram (FBD) programming language

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.

Mathematical functions and other complex functions can be represented directly in conjunction with the logic boxes. To create the logic for complex operations, insert parallel branches between the boxes.



Understanding EN and ENO for the "box" instructions

Both LAD and FBD use "power flow" (EN and ENO) for some "box" instructions. Certain instructions (such as math and move instructions) display parameters for EN and ENO. These parameters relate to power flow and determine whether the instruction is executed during that scan.

- EN (Enable In) is a Boolean input for boxes in LAD and FBD. 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, then box will always be executed.
- ENO (Enable Out) is a Boolean output for boxes in LAD and FBD. 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.

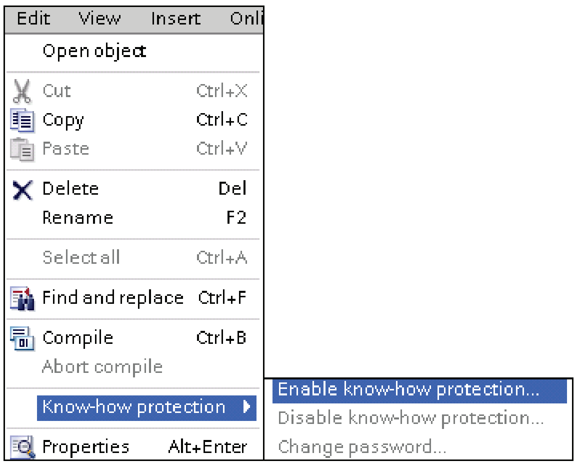
Program editor	Inputs/outputs	Operands	Data type
LAD	EN, ENO	Power flow	BOOL
FBD	EN	I, I:P, Q, M, DB, Temp, Power Flow	BOOL
	ENO	Power Flow	BOOL

5.6 Copy protection

Copy or "know-how" protection allows you to prevent one or more code blocks (OB, FB, or FC) in your program from unauthorized access. You create a password to limit access to the code block.

When you configure a block for "know-how" protection, the code within the block cannot be accessed except after entering the password.

To copy-protect the block, select the "Know how protection" command from the "Edit" menu. You then enter a password that allows access to the 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:

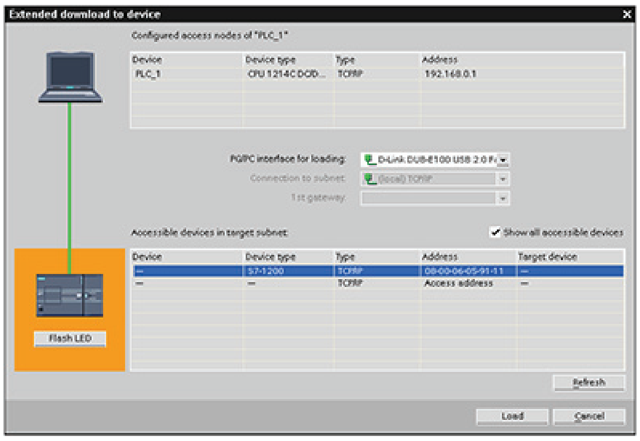
- Block title, block comment, and block properties
- Transfer parameters (IN, OUT, IN_OUT, Return)
- Call structure of the program
- Global tags in the cross references (without information on the point of use), but local tags are hidden

5.7 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.



Configured access nodes of "PLC_1"

Device	Device type	Type	Address
PLC_1	CPU 1214C DC/D...	TCP/IP	192.168.0.1

PG/PC interface for loading: D-Link DUB-E100 USB 2.0 F...
Connection to subnet: (local) TCP/IP
1st gateway:

Accessible devices in target subnet: ✓ Show all accessible devices

Device	Device type	Type	Address	Target device
—	S7-1200	TCP/IP	08-00-06-05-91-11	—
—	—	TCP/IP	Access address	—

5.8 Uploading the elements of your program

You can upload all program blocks and the tag table from an online CPU to an offline project, but you cannot upload the device configuration or watch tables. You cannot upload into an empty project; you must have an offline CPU to be able to upload. You cannot upload a single block; you can only upload the whole program. If an upload is performed, the offline CPU will be "cleared" (all blocks and tag table are deleted) before the upload after a check-question. You cannot edit a block in the online area; you must first upload it to the offline area, then modify it there, and then download it back to the PLC.

There are two ways to perform the upload: drag and drop in the Project tree, or synchronize in the Compare editor.

Drag and drop in the project tree

1. Create a new project.
2. Add a CPU device which matches the CPU you are uploading from.
3. Expand the CPU node once so that the "Program blocks" folder is visible.
4. From the Project tree, expand the node for "Online access", expand the node for the desired network, and double click "Update accessible devices.
5. After the available CPUs are listed, expand the node for the CPU of interest.
6. Left-click and hold the Program blocks folder from the Online access area and drag it up to the Program blocks folder from the offline area, then release the left mouse button. The mouse pointer should change to a ‘+’ when you are over the correct area.
7. You should see the "Upload preview" dialog open. Click in the box for "Continue", and then click "Upload from device".
8. Allow the upload to complete. You should now see all the program blocks, technology blocks, and tags in the offline area.
9. Since the device configuration cannot be uploaded, use Device configuration to manually setup the CPU properties, including the desired IP address, and to add the other devices to the offline project.

You can also drag from the online area to the "Program blocks" area of an existing program. That is, the Program-blocks offline area does not have to be empty. In this case, the existing program will be deleted and replaced by the online program.

Synchronize in the compare editor

1. Open the project that contains the project.
2. In the Project tree, select the offline CPU to compare.
3. Open the "Compare" editor either by right-clicking the offline CPU, or by selecting the "Compare offline/online" command from the "Tools" menu.
4. The Compare editor lists the differences under the "Program blocks" folder. Click the symbol in the action column. To upload the project, select "Upload from device".
5. Click the "Synchronize online and offline" button to copy the project from the online CPU to the offline CPU.

5.9 Debugging and testing the program

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 and outputs (I:P and Q: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.

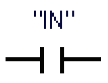
The watch table also allows you to "force" or set a tag to a specific value. For more information about forcing, see the section on forcing values in the CPU (Page 273) in the "Online and Diagnostics" chapter.

6 Programming instructions

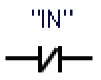
6.1 Basic instructions

6.1.1 Bit logic

LAD contacts



Normally Open



Normally Closed

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.

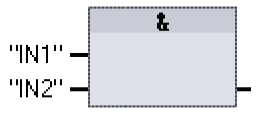
Parameter	Data type	Description
IN	Bool	Assigned bit

- 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.

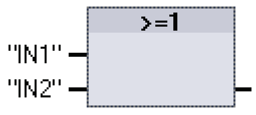
FBD, AND, OR, and XOR boxes

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 binary 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".

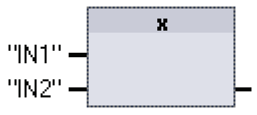
Box inputs and outputs can be connected to another logic box, or you can enter a bit address or bit symbol name for an unconnected input. When the box instruction is executed, the current input states are applied to the binary box logic and, if true, the box output will be true.



AND logic



OR logic



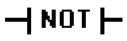
XOR logic

Parameter	Data type	Description
IN1, IN2	Bool	Input bit

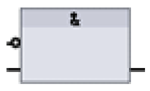
- All inputs of an AND box must be TRUE for the output to be TRUE.
- Any input of an OR box must be TRUE for the output to be TRUE.
- An odd number of the inputs of an XOR box must be TRUE for the output to be TRUE.

NOT logic inverter

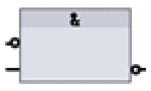
For FBD programming, you can drag the "Negate binary input" tool from the "Favorites" toolbar or instruction tree and then drop it on an input or output to create a logic inverter on that box connector.



LAD: NOT contact inverter



FBD: AND box with one inverted logic input

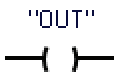


FBD: AND box with inverted logic input and output

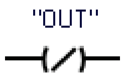
The LAD NOT contact inverts the logical state of power flow input.

- 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.

LAD output coil



Output coil



Inverted output coil

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 S7-1200. 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.

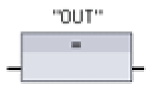
You can specify an immediate write of a physical output using ":P" following the Q offset (example: "%Q3.4:P"). For an immediate write, the bit data values are written to the process image output and directly to physical output.

Parameter	Data type	Description
OUT	Bool	Assigned bit

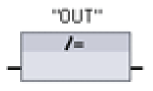
- If there is power flow through an output coil, then the output bit is set to 1.
- If there is no power flow through an output coil, then the output coil bit is set to 0.
- If there is power flow through an inverted output coil, then the output bit is set to 0.
- If there is no power flow through an inverted output coil, then the output bit is set to 1.

FBD output assignment box

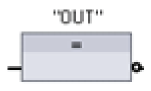
In FBD programming, LAD coils are transformed into assignment (= and /=) boxes where you specify a bit address for the box output. Box inputs and outputs can be connected to other box logic or you can enter a bit address.



Output assignment



Inverted output assignment



Output assignment with inverted output

Parameter	Data type	Description
OUT	Bool	Assigned bit

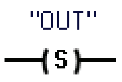
- If the output box input is 1, then the OUT bit is set to 1.
- If the output box input is 0, then the OUT bit is set to 0.
- If the inverted output box input is 1, then the OUT bit is set to 0.
- If the inverted output box input is 0, then the OUT bit is set to 1.

6.1.1.1 Set and reset instructions

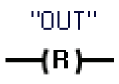
S and R: Set and Reset 1 bit

- 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.
- 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.
- These instructions can be placed anywhere in the network.

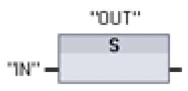
LAD: Set



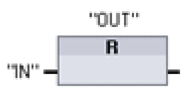
LAD: Reset



FBD: Set



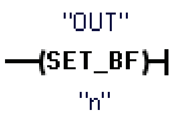
FBD: Reset



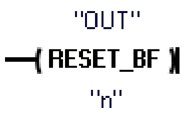
Parameter	Data type	Description
IN (or connect to contact/gate logic)	Bool	Bit location to be monitored
OUT	Bool	Bit location to be set or reset

SET_BF and RESET_BF: Set and Reset Bit Field

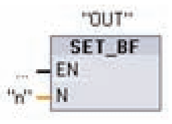
LAD: SET_BF



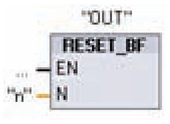
LAD: RESET_BF



FBD: SET_BF



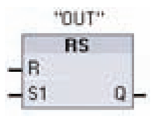
FBD: RESET_BF

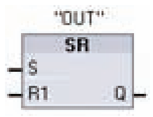


Parameter	Data type	Description
n	Constant	Number of bits to write
OUT	Element of a boolean array	Starting element of a bit field to be set or reset Example: #MyArray[3]

- 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.
- 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.
- These instructions must be the right-most instruction in a branch.

RS and SR: Set-dominant and Reset-dominant bit latches





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.

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.

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.

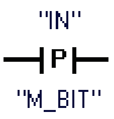
Parameter	Data type	Description
S, S1	BOOL	Set input; 1 indicates dominance
R, R1	BOOL	Reset input; 1 indicates dominance
OUT	BOOL	Assigned bit output "OUT"
Q	BOOL	Follows state of "OUT" bit

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

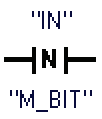
6.1.1.2 Positive and negative edge instructions

Positive and Negative transition detectors

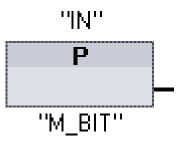
P contact: LAD



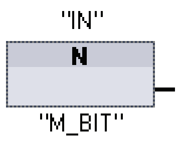
N contact: LAD



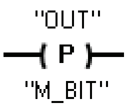
P box: FBD



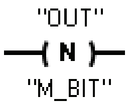
N box: FBD



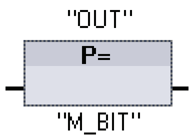
P coil: LAD



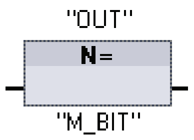
N coil: LAD



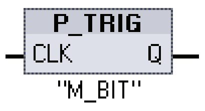
P= box: FBD



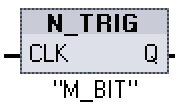
N= box: FBD



P_TRIG: LAD\FBD



N_TRIG: LAD\FBD



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

P contact: 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.
N contact: 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.
P box: 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.
N box: 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.
P coil: 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.
N coil: 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.
P= box: 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.
N= box: 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.
P_TRIG: LAD/FBD	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.
N_TRIG (LAD/FBD)	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 P_TRIG instruction can be located anywhere except the end of a branch.

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.

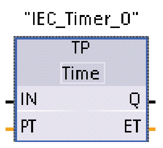
6.1.2 Timers

You use the timer instructions to create programmed time delays:

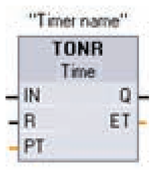
- TP: The Pulse timer generates a pulse with a preset width time.
- TON: The ON-delay timer output Q is set to ON after a preset time delay.
- TOF: The OFF-delay timer output Q is reset to OFF after a preset time delay.
- TONR: The ON-delay Retentive timer output is set to ON after a preset time delay. Elapsed time is accumulated over multiple timing periods until the R input is used to reset the elapsed time.
- RT: Reset a timer by clearing the time data stored in the specified timer instance data block.

Each timer uses a structure stored in a data block to maintain timer data. You assign the data block when the timer instruction is placed in the editor.

When you place timer instructions in a function block, 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 data block.



TP, TON, and TOF timers have the same input and output parameters.



The TONR timer has the additional reset input parameter R.

Create you own "Timer name" that names the timer Data Block and describes the purpose of this timer in your process.

"Timer name"
----[RT]----

The RT instruction resets the timer data for the specified timer.

Parameter	Data type	Description
IN	Bool	Enable timer input
R	Bool	Reset TONR elapsed time to zero
PT	Bool	Preset time value input
Q	Bool	Timer output
ET	Time	Elapsed time value output
Timer data block	DB	Specify which timer to reset with the RT instruction

Parameter IN starts and stops the timers:

- The 0 to 1 transition of parameter IN starts timers TP, TON, and TONR.
- The 1 to 0 transition of parameter IN starts timer TOF.

The following table shows the effect of value changes in the PT and IN parameters.

Timer	Changes in the PT and IN 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.

TIME values

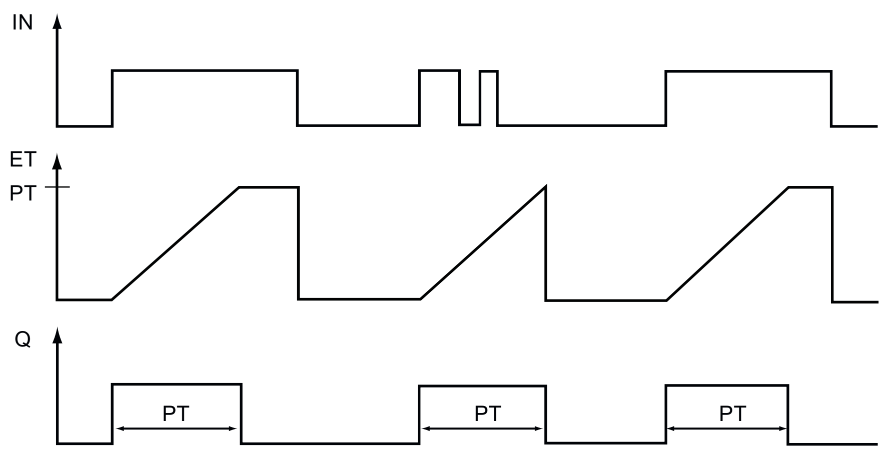
PT (preset time) and ET (elapsed time) values are stored in memory as signed double integers that represent milliseconds of time. TIME data uses the T# identifier and can be entered as a simple time unit "T#200ms" or as compound time units "T#2s_200ms".

Data type	Size	Valid number ranges
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

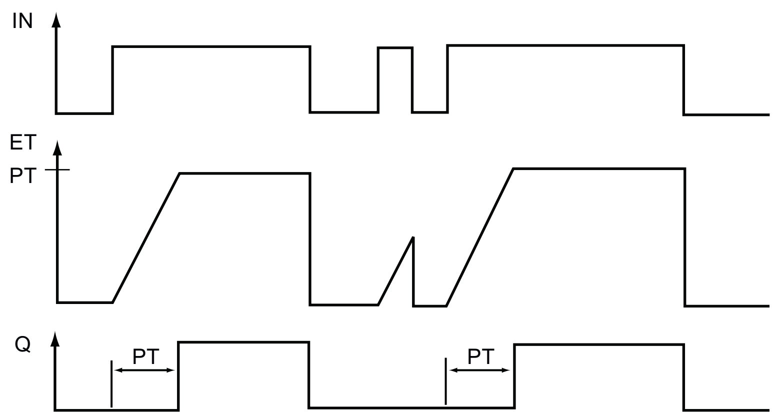
Note

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.

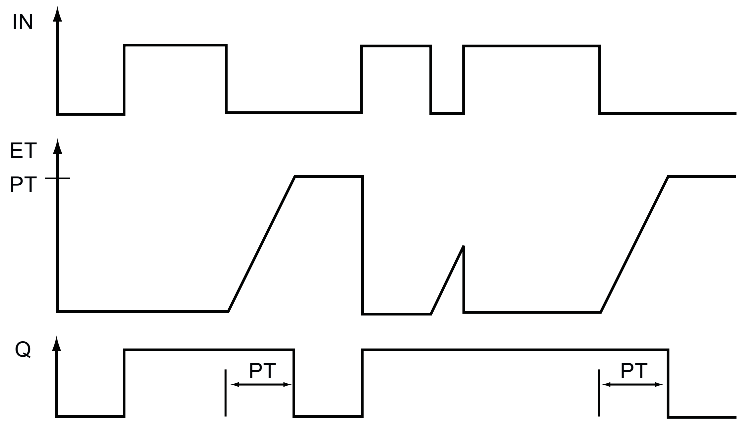
TP:
Pulse timing diagram



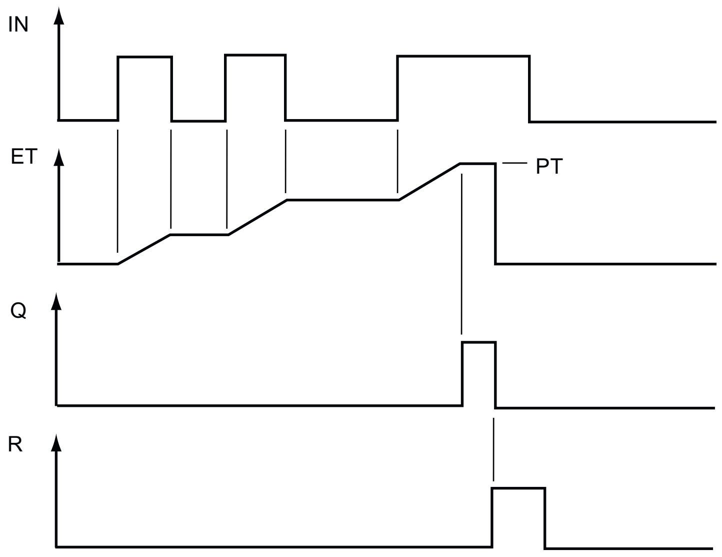
TON:
ON-delay timing diagram



TOF:
OFF-delay timing diagram



TONR:
ON-delay
Retentive
timing
diagram



6.1.3 Counters

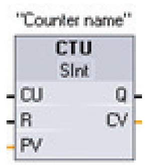
6.1.3.1 Counters

You use the counter instructions to count internal program events and external process events:

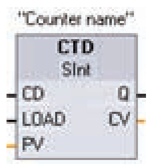
- CTU is a count up counter.
- CTD is a count down counter.
- CTUD is a count up and down counter.

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. 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.

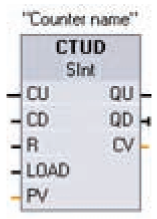
When you place counter instructions in a function block, you can select the Multi-instance data block option, the counter structure names can be different with separate data structures, but the counter data is contained in a single data block and does not require a separate data block for each counter. 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 data block.



Select the count value data type from the drop-down list under the box name.



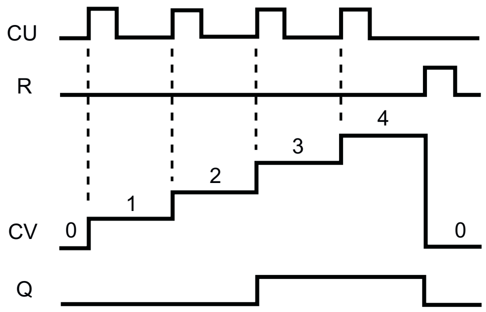
Create your own "Counter name" that names the counter Data Block and describes the purpose of this counter in your process.



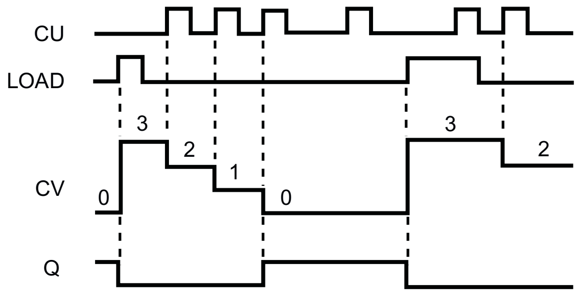
Parameter	Data type	Description
CU, CD	Bool	Count up or count down, by one count
R (CTU, CTUD)	Bool	Reset count value to zero
LOAD (CTD, CTUD)	Bool	Load control for preset value
PV	SInt, Int, DInt, USInt, UInt, UDInt	Preset count value
Q, QU	Bool	True if CV >= PV
QD	Bool	True if CV <= 0
CV	SInt, Int, DInt, USInt, UInt, UDInt	Current count value

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.

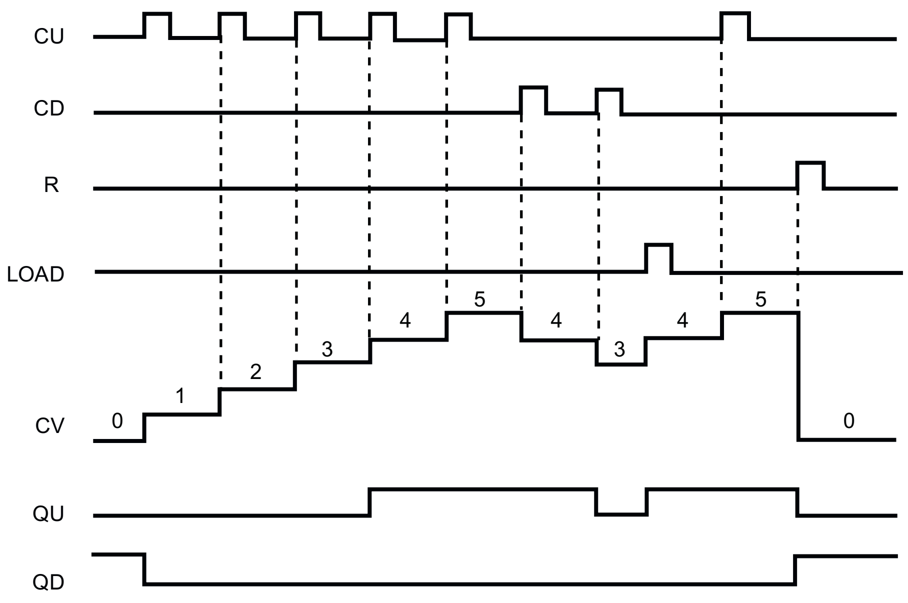
CTU: CTU counts up by 1 when the value of parameter CU changes from 0 to 1. If the value of parameter CV (Current count value) is greater than or equal to the value of parameter PV (Preset count value), then the counter output parameter Q = 1. If the value of the reset parameter R changes from 0 to 1, then the current count value is reset to 0. The following figure shows a CTU timing diagram with an unsigned integer count value (where PV = 3).



CTD: CTD counts down by 1 when the value of parameter CD changes from 0 to 1. 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). The following figure shows a CTD timing diagram with an unsigned integer count value (where PV = 3).



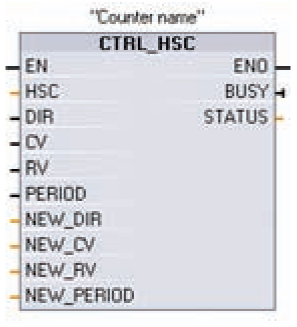
CTUD: CTUD counts up or down by 1 on the 0 to 1 transition of the Count up (CU) or Count down (CD) inputs. If the value of parameter CV (Current count value) is equal to or greater than the value of parameter PV (Preset value), then the counter output parameter QU = 1. If the value of parameter CV is 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 (Preset value) is loaded to the counter as the new CV (Current count value). If the value of the reset parameter R is changes from 0 to 1, the current count value is reset to 0. The following figure shows a CTUD timing diagram with an unsigned integer count value (where PV = 4).



6.1.3.2 CTRL_HSC instruction

The CTRL_HSC instruction controls the high-speed counters that are used to count events that occur faster than the OB execution rate. The counting rate of the CTU, CTD, and CTUD counter instructions is limited by the execution rate of the OB in which they are placed. Refer to the CPU technical specifications (Page 284) for the HSC maximum clock input rates.

A typical use for high-speed counters is to count pulses generated by a motion control shaft encoder.



Each CTRL_HSC instruction uses a structure stored in a data block to maintain data. You assign the data block when the CTRL_HSC instruction is placed in the editor.

Create you own "Counter name" that names the counter Data Block and describes the purpose of this counter in your process.

Parameter	Parameter type	Data type	Description
HSC	IN	HW_HSC	HSC identifier
DIR	IN	Bool	1 = Request new direction
CV	IN	Bool	1 = Request to set new counter value
RV	IN	Bool	1= Request to set new reference value
PERIOD	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: .01, .1, or 1 (only for frequency measurement mode)
BUSY	OUT	Bool	Function busy
STATUS	OUT	Word	Execution condition code

You must configure the high-speed counters in the project settings PLC device configuration before you can use high-speed counters in your program. The HSC device configuration settings select counting modes, I/O connections, interrupt assignment, and operation as a high-speed counter or as a device to measure pulse frequency. You can operate the high-speed counter with no program control or with program control.

Many high-speed counter configuration parameters are set only in the project device configuration. Some high-speed counter parameters are initialized in the project device configuration, but can be modified later under program control.

The CTRL_HSC instruction parameters 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 usually 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 high-speed counter hardware configuration. You may use program logic to directly read the count value and 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.

CTRL_HSC parameter details:

- If an update of a parameter value is not requested, then the corresponding input values are ignored.
- 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.
- For a S7-1200 HSC on the CPU or on the Signal Board, the BUSY parameter always has a value of 0.

Condition codes: In the case of an error, ENO is set to 0, and the STATUS output contains a condition code.

STATUS value (W#16#...)	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

6.1.3.3 Operation of the high-speed counter

A high-speed counter (HSC) can be used 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.)

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.

Description			Default Input Assignment			Function
HSC	HSC1	Built In or Signal Board or monitor PTO 0[1]	I0.0 I4.0 PTO 0 Pulse	I0.1 I4.1 PTO 0 Direction	I0.3 I4.3 -	
	HSC:	Built In or signal board or monitor PTO 1[1]	I0.2 I4.2 PTO 1 Pulse	I0.3 I4.3 PTO 1 Direction	I0.1 I4.1 -	
	HSC3[2]	Built In	I0.4	I0.5	I0.7	
	HSC4[3]	Built In	I0.6	I0.7	I0.5	
	HSC5[4]	Built In or Signal Board	I1.0 I4.0	I1.1 I4.1	I1.2 I4.3	
	HSC6[4]	Built In or signal board	I1.3 I4.2	I1.4 I4.3	I1.5 I4.1	
Mode	Single-phase counter with internal direction control		Clock	-	-	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
					Phase Z	Count
	Monitor pulse train outputs (PTO)[1]		Clock	Direction	-	Count

1 Pulse train output monitoring always uses clock and direction. If the corresponding PTO output is configured for pulse only, then the direction output should generally be set for positive counting.

2 HSC3 with a reset input is not possible for the CPU 1211C which supports only 6 built-in inputs.

3 HSC4 is not possible for the CPU 1211C which supports only 6 built-in inputs.

4 HSC5 and HSC6 are only supported by the CPU 1211C and CPU 1212C when a signal board is installed.

Accessing the current value for the HSC

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.

High-speed counter	Data type	Default address
HSC1	DInt	ID1000
HSC2	DInt	ID1004
HSC3	DInt	ID1008
HSC4	DInt	ID1012
HSC5	DInt	ID1016
HSC6	DInt	ID1020

Digital I/O points assigned to HSC devices cannot be forced

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 Watch table force function.

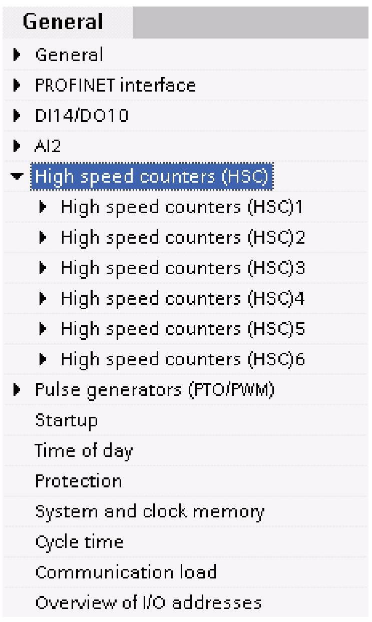
6.1.3.4 Configuration of the HSC

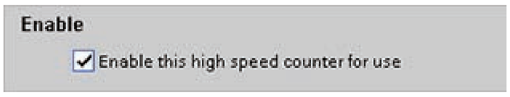
The CPU allows you to configure up to 6 high-speed counters. You edit the "Properties" of the CPU to configure the parameters of each individual HSC.

Configure the parameters for the high-speed counters by editing the "Properties" of the CPU.

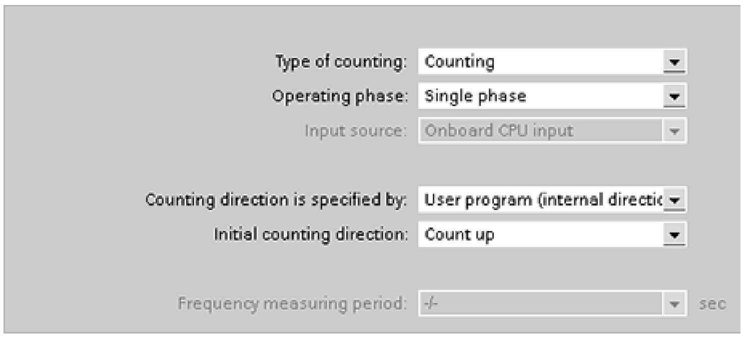
After enabling the HSC, configure the other parameters, such as counter function, initial values, reset options and interrupt events.

After configuring the HSC, you use the CTRL_HSC instruction in your user program to control the operation of the HSC.

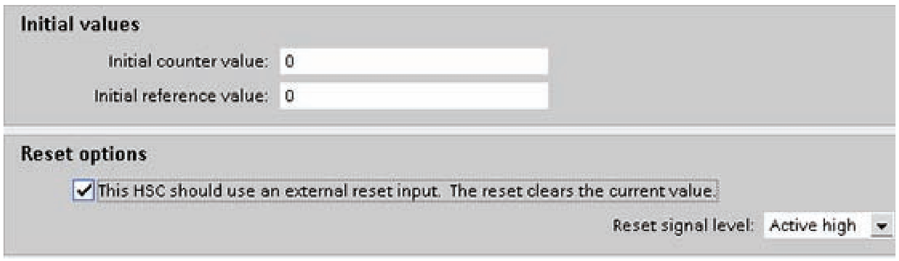

General
- General
- PROFINET interface
- DI14/DO10
- AI2
- High speed counters (HSC)
  - High speed counters (HSC)1
  - High speed counters (HSC)2
  - High speed counters (HSC)3
  - High speed counters (HSC)4
  - High speed counters (HSC)5
  - High speed counters (HSC)6
- Pulse generators (PTO/PWM)
- Startup
- Time of day
- Protection
- System and clock memory
- Cycle time
- Communication load
- Overview of I/O addresses


Enable

✓ Enable this high speed counter for use



Type of counting:	Counting
Operating phase:	Single phase
Input source:	Onboard CPU input
Counting direction is specified by:	User program (internal directic
Initial counting direction:	Count up
Frequency measuring period:	-/- sec



Initial values

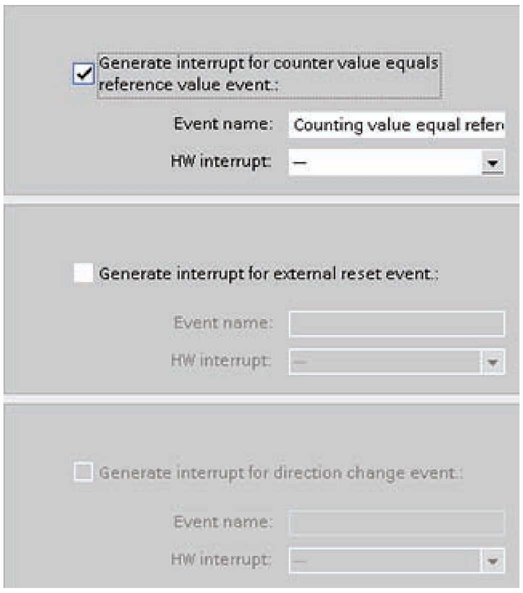
Initial counter value: 0

Initial reference value: 0

Reset options

[✓] This HSC should use an external reset input. The reset clears the current value.

Reset signal level: Active high



[✓] Generate interrupt for counter value equals reference value event.:

Event name: Counting value equal refer

HW interrupt: —

[ ] Generate interrupt for external reset event.:

Event name:

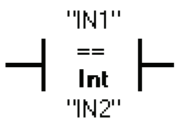
HW interrupt: —

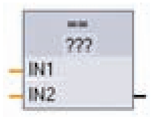
[ ] Generate interrupt for direction change event.:

Event name:

HW interrupt: —

6.1.4 Compare





You use the compare instructions to compare 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.

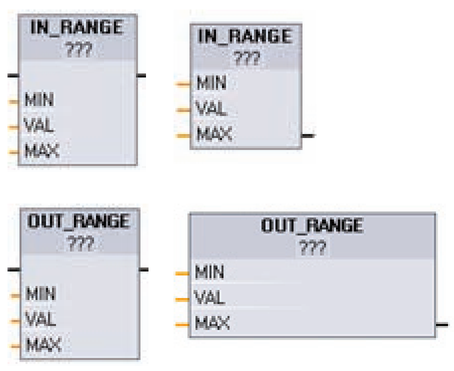
LAD FBD

After you click on the instruction in the program editor, you can select the comparison type and data type from the drop-down menus.

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

Parameter	Data type	Description
IN1, IN2	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, String, Char, Time, DTL, Constant	Values to compare

In Range and Out of Range instructions



LAD FBD

You use the IN_RANGE and OUT_RANGE instructions to test whether an input value is in or out of a specified value range. If the comparison is TRUE, then the box output is TRUE.

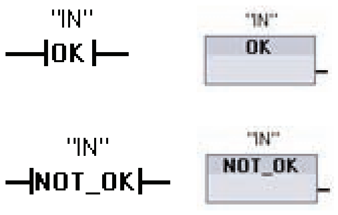
The input parameters MIN, VAL, and MAX must be the same data type.

After you click on the instruction in the program editor, you can select the data type from the dropdown menus.

Relation type	The comparision is TRUE if:
IN_RANGE	MIN <= VAL <= MAX
OUT_RANGE	VAL < MIN or VAL > MAX

Parameter	Data type	Description
MIN, VAL, MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real, Constant	Comparator inputs

OK and Not OK instructions



You use the OK and NOT_OK instructions to test whether an input data reference is a valid real number according to IEEE specification 754. 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.

LAD FBD

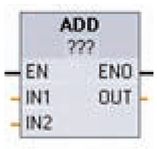
A Real or LReal value is invalid if it is +/- INF (infinity), NaN (Not a Number), or if it is a denormalized value. A denormalized value is a number very close to zero. The CPU substitutes a zero for a denormalized value in calculations.

Instruction	The Real number test is TRUE if:
OK	The input value is a valid Real number
NOT_OK	The input value is not a valid Real number

Parameter	Data type	Description
IN	Real, LReal	Input data

6.1.5 Math

Add, subtract, multiply and divide instructions



You use a math box instruction to program the basic mathematical operations:

- 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.

Click below the box name and select a data type from the drop-down menu.

Note

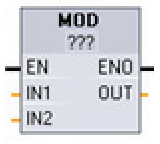
The basic math instruction parameters IN1, IN2, and OUT must be the same data type.

Parameter	Data type	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

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.

ENO status	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.

6.1.5.1 MOD instruction



You use a MOD (modulo) instruction for the IN1 modulo IN2 math operation. The operation IN1 MOD IN2 = IN1 - (IN1 / IN2) = parameter OUT.

Click below the box name and select a data type from the drop-down menu.

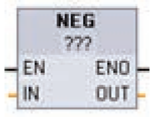
Note

The IN1, IN2, and OUT parameters must be the same data type.

Parameter	Data type	Description
IN1 and IN2	Int, DInt, USInt, UInt, UDInt, Constant	Modulo inputs
OUT	Int, DInt, USInt, UInt, UDInt	Modulo output

ENO status	Description
1	No error
0	Value IN2 = 0, OUT is assigned the value zero

NEG instruction



You use the NEG (negation) instruction to invert the arithmetic sign of the value at parameter IN and store the result in parameter OUT.

Click below the box name and select a data type from the drop-down menu.

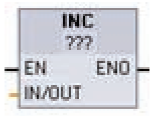
Note

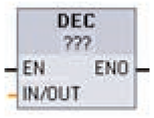
The IN and OUT parameters must be the same data type.

Parameter	Data type	Description
IN	SInt, Int, DInt, Real, LReal, Constant	Math operation input
OUT	SInt, Int, DInt, Real, LReal	Math operation output

ENO status	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.

Increment and Decrement instructions





You use the INC and DEC instructions to:

- Increment a signed or unsigned integer number value

  INC (increment): Parameter IN/OUT value +1 = parameter IN/OUT value
- Decrement a signed or unsigned integer number value

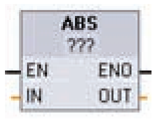
  DEC (decrement): Parameter IN/OUT value - 1 = parameter IN/OUT value

Click below the box name and select a data type from the drop-down menu.

Parameter	Data type	Description
IN/OUT	SInt, Int, DInt, USInt, UInt, UDInt	Math operation input and output

ENO status	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.

Absolute Value instruction



You use the ABS instruction to get the absolute value of a signed integer or real number at parameter IN and store the result in parameter OUT.

Click below the box name and select a data type from the drop-down menu.

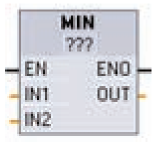
Note

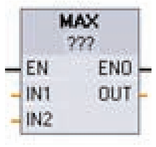
The IN and OUT parameters must be the same data type.

Parameter	Data type	Description
IN	SInt, Int, DInt, Real, LReal	Math operation input
OUT	SInt, Int, DInt, Real, LReal	Math operation output

ENO status	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.

MIN and MAX instructions





You use the MIN (minimum) and MAX (maximum) instructions as follows:

- MIN compares the value of two parameters IN1 and IN2 and assigns the minimum (lesser) value to parameter OUT.
- MAX compares the value of two parameters IN1 and IN2 and assigns the maximum (greater) value to parameter OUT.

Click below the box name and select a data type from the drop-down menu.

Note

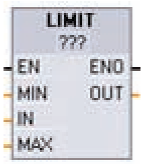
The IN1, IN2, and OUT parameters must be the same data type.

Parameter	Data type	Description
IN1, IN2	SInt, Int, DInt, USInt, UInt, UDInt, Real, Constant	Math operation inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real	Math operation output

ENO status	Description
1	No error
0	For Real data type only: • One or both inputs is not a Real number (NaN). • The resulting OUT is +/- INF (infinity).

Limit instruction

You use the Limit instruction to test if the value of parameter IN is inside the value range specified by parameters MIN and MAX. The OUT value is clamped at the MIN or MAX value, if the IN value is outside this range.



- If the value of parameter IN is inside 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).

Click below the box name and select a data type from the drop-down menu.

Note

The MIN, IN, MAX, and OUT parameters must be the same data type.

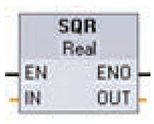
Parameter	Data type	Description
MIN, IN, and MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real, Constant	Math operation inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real	Math operation output

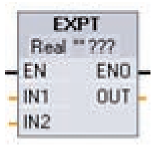
ENO status	Description
1	No error
0	Real: If one or more of the values for MIN, IN and MAX is NaN (Not a Number), then NaN is returned.
0	If MIN is greater than MAX, the value IN is assigned to OUT.

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
- SIN: Sine (sin(IN radians) = OUT)
- COS: Cosine (cos(IN radians) = OUT)
- TAN: Tangent (tan(IN radians) = OUT)
- ASIN: Inverse sine (arcsine(IN) = OUT radians), where the sin(OUT radians) = IN
- ACOS: Inverse cosine (arccos(IN) = OUT radians), where the cos(OUT radians) = IN
- ATAN: Inverse tangent (arctan(IN) = OUT radians), where the tan(OUT radians) = IN
- FRAC: Fraction (fractional part of floating point number IN = OUT)
- EXPT: General exponential (IN1 ^{IN2} = OUT)





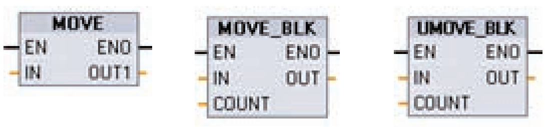
Click below the box name and select a data type from the drop-down menu. EXPT parameters IN1 and OUT are always Real. You can select the data type for the exponent parameter IN2.

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

ENO status	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
		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

6.1.6 Move

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.

- MOVE: Copies a data element stored at a specified address to a new address
- MOVE_BLK: Interruptible move that copies a block of data elements to a new address
- UMOVE_BLK: Uninterruptible move that copies a block of data elements to a new address

MOVE		
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

MOVE_BLK, UMOVE_BLK		
Parameter	**Data type**	**Description**
IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, Byte, Word, DWord	Source start address
COUNT	UInt	Number of data elements to copy
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, 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)
- 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_CONV
- 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.

The MOVE instruction copies a single data element from the source address specified by the IN parameter to the destination address 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.

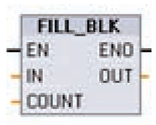
MOVE_BLK and UMOVE_BLK instructions differ in how interrupts are handled:

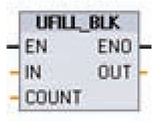
- Interrupt events are **queued and processed** during MOVE_BLK execution. Use the MOVE_BLK instruction when the data at the move destination address is not used within an interrupt OB subprogram or, if used, the destination data does not have to be consistent. If a MOVE_BLK operation is interrupted, 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 86).

ENO is always true following execution of the MOVE instruction.

ENO status	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.

Fill instructions





You use the FILL_BLK and UFILL_BLK instructions as follows:

- FILL_BLK: The interruptible fill instruction fills an address range with copies of a specified data element.
- UFILL_BLK: The uninterruptible fill instruction fills an address range with copies of a specified data element.

Parameter	Data type	Description
IN	SInt, Int, DIntT, USInt, UInt, UDInt, Real, BYTE, Word, DWord	Data source address
COUNT	USInt, UInt	Number of data elements to copy
OUT	SInt, Int, DIntT, USInt, UInt, UDInt, Real, 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.

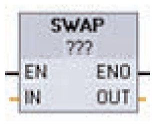
The FILL_BLK and UFILL_BLK instructions copy the source data element IN to the destination where the initial address is specified by the parameter OUT. The copy process repeats and a block of adjacent addresses is filled until the number of copies is equal to the COUNT parameter.

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.

ENO status	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.

6.1.6.1 Swap instruction



You use the SWAP instruction to reverse 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.

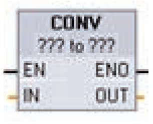
Click below the box name and select a data type from the drop menu.

Parameter	Data type	Description
IN	Word, DWord	Ordered data bytes IN
OUT	Word, DWord	Reverse ordered data bytes OUT

	Example: Parameter IN = MB0 Pre SWAP execution				Example: Parameter OUT = MB4, Post SWAP execution			
Address	MB0	MB1			MB4	MB5		
W#16#1234	12	34			34	12		
WORD	MSB	LSB			MSB	LSB		
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

6.1.7 Convert

Convert instruction



You use the CONVERT instruction to convert a data element from one data type to another data type. Click below the box name and then select IN and OUT data types from the dropdown list.

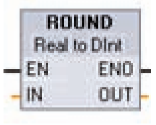
After you select the (convert from) data type, a list of possible conversions is shown in the (convert to) dropdown list. Conversions from and to BCD16 are restricted to the Int data type. Conversions from and to BCD32 are restricted to the DInt data type.

Click below the box name and select data types from the drop-down menus.

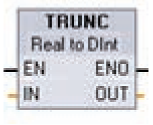
Parameter	Data type	Description
IN	SInt, Int, DInt, USInt, UInt, UDInt, Byte, Word, DWord, Real, LReal, Bcd16, Bcd32	IN value
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Byte, Word, DWord, Real, LReal, Bcd16, Bcd32	IN value converted to a new data type

ENO status	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

Round and Truncate instructions



ROUND converts a real number to an integer. The real number fraction is rounded to the nearest integer value (IEEE - round to nearest). If the Real number is exactly one-half the span between two integers (i.e. 10.5), then the Real number is rounded to the even integer. For example, ROUND (10.5) = 10 or ROUND (11.5) = 12.

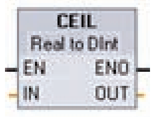


TRUNC converts a real number to an integer. The fractional part of the real number is truncated to zero (IEEE - round to zero).

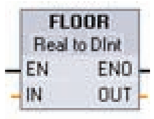
Parameter	Data type	Description
IN	Real, LReal	Floating point input
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Rounded or truncated output

ENO status	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN

Ceiling and Floor instructions



CEIL converts a real number to the smallest integer greater than or equal to that real number (IEEE - round to +infinity).



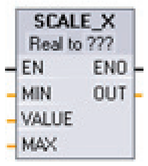
FLOOR converts a real number to the greatest integer smaller than or equal to that real number (IEEE - round to -infinity).

Parameter	Data type	Description
IN	Real, LReal	Floating point input
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Converted output

ENO status	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN

6.1.7.1 Scale and normalize instructions

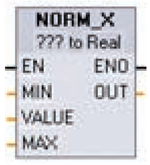
Scale and normalize instructions



SCALE_X 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

For SCALE_X, parameters MIN, MAX, and OUT must be the same data type.



NORM_X 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)

For NORM_X, parameters MIN, VALUE, and MAX must be the same data type.

Click below the box name and select a data type from the drop-down menu.

Parameter	Data type	Description
MIN	SInt, Int, DInt, USInt, UInt, UDInt, Real	Input minimum value for range
VALUE	SCALE_X: Real NORM_X: SInt, Int, DInt, USInt, UInt, UDInt, Real	Input value to scale or normalize
MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real	Input maximum value for range
OUT	SCALE_X: SInt, Int, DInt, USInt, UInt, UDInt, Real NORM_X: Real	Scaled or Normalized output value

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)

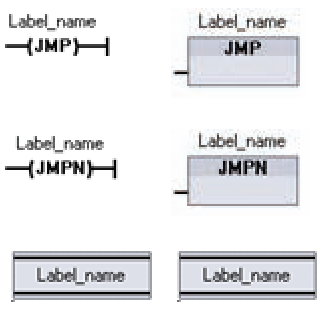
If parameter VALUE is less than MIN or greater than MAX, the linear scaling operation can produce normalized OUT values that are less than 0.0 or greater than 1.0. NORM_X execution sets ENO = TRUE in this case.

ENO status	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

6.1.8 Program control

Jump and label instructions

You use program control instructions for conditional control of the execution sequence:



JMP: 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.

JMPN: If there is no power flow to a JMP coil (LAD), or if the JMP box input is false (FBD), then program execution continues with the first instruction following the specified label.

Label: Destination label for a JMP or JMPN jump instruction.

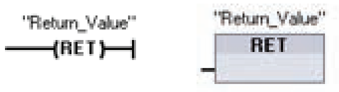
LAD FBD

Parameter	Data type	Description
Label_name	Label identifier	Identifier for Jump instructions and the corresponding jump destination program label

You create your label names by typing in the LABEL instruction directly. The available label names for the JMP and JMPN label name field can be selected using the parameter helper icon. You can also type a label name directly into the JMP or JMPN instruction.

Return_Value (RET) execution control instruction

LAD FBD



You use the RET instruction to terminate the execution of the current block.

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.

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.

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:

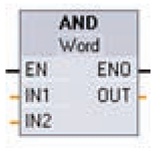
     TRUE, FALSE, or a memory location that specifies the required return value.
   - Add more instructions.
3. Call the FC from MAIN [OB1].

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.

6.1.9 Logical operations

AND, OR, and XOR instructions



AND: Logical AND for BYTE, WORD, and DWORD data types

OR: Logical OR for BYTE, WORD, and DWORD data types

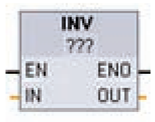
XOR: Logical exclusive OR for BYTE, WORD, and DWORD data types

Click below the box name and select a data type from the drop menu.

Parameter	Data type	Description
IN1, IN2	Byte, Word, DWord	Logical inputs
OUT	Byte, Word, DWord	Logical output

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.

Invert instruction

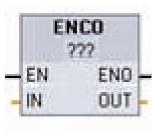


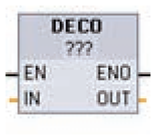
You use the INV instruction to get the binary one's complement of the parameter IN. The one's complement is formed by inverting each bit value of the IN parameter (changing each 0 to 1 and each 1 to 0). ENO is always TRUE following the execution of this instruction.

Click below the box name and select a data type from the drop-down menu.

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

Encode and decode instructions





ENCO encodes a bit pattern to a binary number.

DECO decodes a binary number to a bit pattern.

Click below the box name and select a data type from the drop-down menu.

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

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 OUT. If the parameter IN value is 0000 0000, then ENO is set to FALSE.

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.

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.

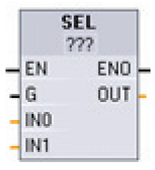
DECO parameter IN range:

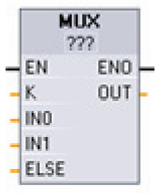
- 3 bits (values 0-7) IN are used to set 1 bit position in a byte OUT
- 4-bits (values 0-15) IN are used to set 1 bit position in a word OUT
- 5 bits (values 0-31) IN are used to set 1 bit position in a double word OUT

DECO IN value		DECO OUT value (Decode single bit position)
		Byte OUT (8 bits):
Min. IN	0	00000001
Max. IN	7	10000000
		Word OUT (16 bits):
Min. IN	0	0000000000000001
Max. IN	15	1000000000000000
		DWord OUT: (32 bits):
Min. IN	0	00000000000000000000000000000001
Max. IN	31	10000000000000000000000000000000

ENO status	Condition	Result (OUT)
1	No error	Valid bit number
0	IN is zero	OUT is set to zero

Select (SEL) and Multiplex (MUX) instructions





- SEL assigns one of two input values to parameter OUT, depending on the parameter G value.
- MUX assigns one of many input values to parameter OUT, depending on the parameter K value. If the parameter K value exceeds the valid range, the parameter ELSE value is assigned to parameter OUT.

Click below the box name and select a data type from the drop-down menu.

SEL	Data type	Description
G	Bool	Selector switch: • FALSE for IN0 • TRUE for IN1
IN0, IN1	SInt, Int, DInt, USInt, UInt, UDInt, Real, Byte, Word, DWord, Time, Char	Inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, Byte, Word, DWord, Time, Char	Output

MUX	Data type	Description
K	UInt	Selector value: • 0 for IN0 • 1 for IN1 • ...
IN0, IN1, ...	SInt, Int, DInt, USInt, UInt, UDInt, Real, Byte, Word, DWord, Time, Char	Inputs
ELSE	SInt, Int, DInt, USInt, UInt, UDInt, Real, Byte, Word, DWord, Time, Char	Input substitute value (optional)
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, Byte, Word, DWord, Time, Char	Output

Input variables and the output variable must be of the same data type.

- The SEL instruction always selects between two IN values.
- The MUX instruction has two IN parameters when first placed in the program editor, but it can be expanded to add more IN parameters.

Use the following methods to add and remove input parameters for the MUX instruction:

- To add an input, 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.

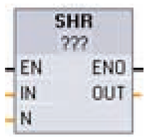
Condition codes: ENO is always TRUE following execution of the SEL instruction.

ENO status (MUX)	MUX condition	MUX result OUT
1	No error	Selected IN value is assigned to OUT

0	K is greater than or equal to the number of IN parameters	No ELSE provided:
		OUT is unchanged
		ELSE provided:
		ELSE value assigned to OUT

6.1.10 Shift and Rotate

Shift instruction



You use the shift instructions 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

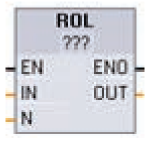
Click below the box name and select a data type from the drop-down list.

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 and 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.

SHL example for Word size data: Shift in zeros from the left			
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

Rotate instruction



You use the rotate instructions 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

Click below the box name and select a data type from the drop-down menu.

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 and 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.

ROR example for WORD size data: Rotate bits out the right -side into the left -side			
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

6.2 Extended instructions

6.2.1 Common error parameters for extended instructions

The extended instruction descriptions describe run-time errors that can occur for each program instruction. In addition to these errors, the common errors listed below are also possible. When a code block is executed and one of the common errors occurs, then the CPU will go to STOP mode unless you use the GetError or GetErrorID instructions within that code block to create a programmed reaction to the error.

Condition code value (W#16#....)	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

6.2.2 Clock and calendar instructions

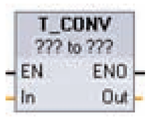
Date and Time instructions

You 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

Data type	Size (bits)	Valid ranges
Time	32 Stored as	T#-24d_20h_31m_23s_648ms to T#24d_20h_31m_23s_647ms -2,147,483,648 ms to +2,147,483,647 ms
DTL data structure		
Year: UInt	16	1970 to 2554
Month: USInt	8	1 to 12
Day: USInt	8	1 to 31
Weekday: USInt	8	1=Sunday to 7=Saturday
Hour: USInt	8	0 to 23

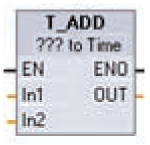
Data type	Size (bits)	Valid ranges
Minute: USInt	8	0 to 59
Second: USInt	8	0 to 59
Nanoseconds: UDInt	32	0 to 999,999,999



T_CONV (Time Convert) converts a Time data type to a DInt data type, or the reverse conversion from DInt data type to Time data type.

Parameter	Parameter type	Data type	Description
IN	IN	DInt, Time	Input Time value or Dint value
OUT	OUT	DInt, Time	Converted DInt value or Time value

Select the IN and OUT data types from the drop-down lists available below the instruction name.



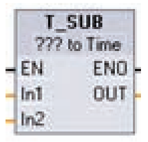
T_ADD (Time Add) adds the input IN1 value (DTL or Time data types) with the input IN2 Time value. Parameter OUT provides the DTL or Time value result.

Two data type operations are possible:

- Time + Time = Time
- DTL + Time = DTL

Parameter	Parameter type	Data type	Description
IN1	IN	DTL, Time	DTL or Time value
IN2	IN	Time	Time value to add
OUT	OUT	DTL, Time	DTL or Time sum

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.



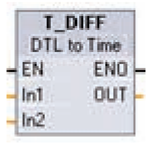
T_SUB (Time Subtract) subtracts the IN2 Time value from IN1 (DTL or Time value). Parameter OUT provides the difference value as a DTL or Time data type.

Two data type operations are possible:

- Time - Time = Time
- DTL - Time = DTL

Parameter	Parameter type	Data type	Description
IN1	IN	DTL, Time	DTL or Time value
IN2	IN	Time	Time value to subtract
OUT	OUT	DTL, Time	DTL or Time difference

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.



T_DIFF (Time Difference) subtracts the IN2 DTL value from IN1 DTL value. Parameter OUT provides the difference value as a Time data type.

- DTL - DTL = Time

Parameter	Parameter type	Data type	Description
IN1	IN	DTL	DTL value
IN2	IN	DTL	DTL value to subtract
OUT	OUT	Time	Time difference

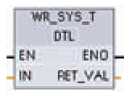
Condition codes: ENO = 1 means no error occurred. ENO = 0 and parameter OUT = 0 errors:

- Invalid DTL value
- Invalid Time value

Clock instructions

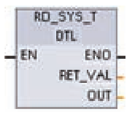
You use the clock instructions to set and read the PLC system clock. The data type DTL is used to provide date and time values.

DTL structure	Size	Valid ranges
Year: UInt	16 bits	1970 to 2554
Month: USInt	8 bits	1 to 12
Day: USInt	8 bits	1 to 31
Weekday: USInt	8 bits	1=Sunday to 7=Saturday
Hour: USInt	8 bits	0 to 23
Minute: USInt	8 bits	0 to 59
Second: USInt	8 bits	0 to 59
Nanoseconds: UDInt	32 bits	0 to 999,999,999



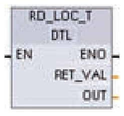
WR_SYS_T (Write System Time) sets the PLC time of day clock with a DTL value at parameter IN. This time value does not include local time zone or daylight saving time offsets.

Parameter	Parameter type	Data type	Description
IN	IN	DTL	Time of day to set in the PLC system clock
RET_VAL	OUT	Int	Execution condition code



RD_SYS_T (Read System Time) reads the current system time from the PLC. This time value does not include local time zone or daylight saving time offsets.

Parameter	Parameter type	Data type	Description
RET_VAL	OUT	Int	Execution condition code
OUT	OUT	DTL	Current PLC system time



RD_LOC_T (Read Local Time) provides the current local time of the PLC as a DTL data type.

Parameter	Parameter type	Data type	Description
RET_VAL	OUT	Int	Execution condition code
OUT	OUT	DTL	Local time

- The local time is calculated by using the time zone and daylight saving time offsets that you set in the CPU Clock device configuration.
- Time zone configuration is an offset to Coordinated Universal Time (UTC) system 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.

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.

RET_VAL (W#16#....)	Description
0000	No error
8080	Local time not available
8081	Illegal year value
8082	Illegal month value

RET_VAL (W#16#....)	Description
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

6.2.3 String and character instructions

6.2.3.1 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

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.

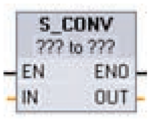
For more information see: Format of the String data type (Page 57)

6.2.3.2 String conversion instructions

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 Convert) 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.

Select the parameter data types from the drop-down lists.

S_CONV (String to value conversions)

Parameter	Parameter type	Data type	Description
IN	IN	String	Input character string
OUT	OUT	String, SInt, Int, DInt, USInt, UInt, UDInt, Real	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.
- Only fixed-point representation is supported. The characters "e" and "E" are not recognized as exponential notation.

S_CONV (Value to string conversions)

Parameter	Parameter type	Data type	Description
IN	IN	String, SInt, Int, DInt, USInt, UInt, UDInt, Real	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.

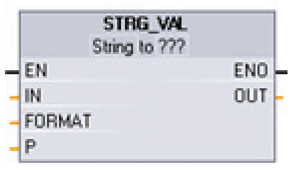
The following table shows the maximum possible string lengths required for each supported data type.

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



STRG_VAL (String to Value) converts a number character string to the corresponding integer or floating point representation. 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.

Parameter P is also returned as an offset count in the original string at the position where the conversion terminated. String data must be initialized before execution as a valid string in memory.

Parameter	Parameter type	Data type	Description
IN	IN	String	The ASCII character string to convert
FORMAT	IN	Word	Output format options
P	IN_OUT	UInt	IN: Index to the first character to be converted (first character = 1) OUT: Index to the next character after conversion process ends
OUT	OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real	Converted number value

STRG_VAL FORMAT parameter

The FORMAT parameter for the STRG_VAL instruction is defined below. The unused bit positions must be set to zero.

Bit 16							Bit 8	Bit 7							Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	f	r

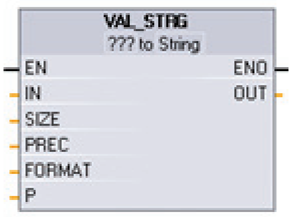
f = Notation format	1= Exponential notation 0 = Fixed point notation
r = Decimal point format	1 = "," (comma character) 0 = "." (period character)

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



VAL_STRG (Value to String) converts an integer, unsigned integer, or floating point value to the corresponding character string representation. 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	Parameter type	Data type	Description
IN	IN	SInt, Int, DInt, USInt, UInt, UDInt, Real	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_OUT	UInt	IN: Index to the first OUT string character to be replaced (first character = 1) OUT: Index to the next OUT string character after replacement
OUT	OUT	String	The converted string

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.

VAL_STRG FORMAT parameter

The FORMAT parameter for the VAL_STRG instruction is defined below. The unused bit positions must be set to zero.

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)

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.

ENO status	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

Examples of S_CONV string to value conversion

IN string	OUT data type	OUT value	ENO
"123"	Int/DInt	123	TRUE
"-00456"	Int/DInt	-456	TRUE
"123.45"	Int/DInt	123	TRUE
"+2345"	Int/DInt	2345	TRUE
"00123AB"	Int/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

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

Examples of STRG_VAL conversion

IN string	FORMAT (W#16#....)	OUT data type	OUT value	ENO
"123"	0000	Int/DInt	123	TRUE
"-00456"	0000	Int/DInt	-456	TRUE
"123.45"	0000	Int/DInt	123	TRUE
"+2345"	0000	Int/DInt	2345	TRUE
"00123AB"	0000	Int/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

Examples of VAL_STRG conversion

The examples are based on an OUT string initialized as follows:

"`Current Temp = xxxxxxxxxx C`"
The "`x`"character represents space characters allocated for the converted value.

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

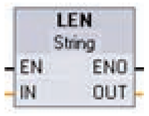
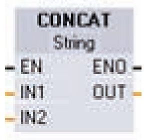
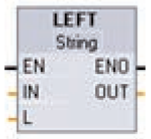
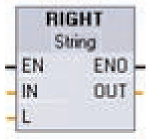
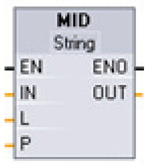
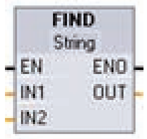
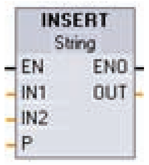
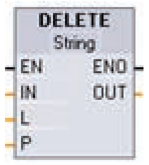
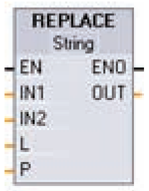
6.2.3.3 String operation instructions

Your control program can use the following string and character instructions to create messages for operator display and process logs.

Common errors for all String operations

String operation instructions that are executed with the illegal or invalid String conditions shown below result in an ENO = 0 and a null string output. Error conditions that occur for a specific instruction are listed below the instruction operation description.

ENO	Condition	OUT
0	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)	

	LEN: Get string length		CONCAT: Concatenate two strings
	LEFT: Get left substring from string		RIGHT: Get right substring form string
	MID: Get middle substring from string		FIND: Find substring or character in string
	INSERT: Insert substring in string		DELETE: Delete substring from string
	REPLACE: Replace substring in string		

LEN instruction

Parameter	Parameter type	Data type	Description
IN	IN	String	Input string
OUT	OUT	UInt	Number of valid characters of IN string

LEN (Length of string) gives the current length of the string IN at output OUT. An empty string has a length of zero. The following table shows the condition codes for the instruction.

ENO	Condition	OUT
1	No invalid string condition	Valid string length

CONCAT instruction

Parameter	Parameter 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)

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. The following table shows the condition codes for the instruction.

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

LEFT instruction

Parameter	Parameter type	Data type	Description
IN	IN	String	Input string
L	IN	Int	Length of the substring to be created, using the left-most L characters of the IN string
OUT	OUT	String	Output string

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.

The following table shows the condition codes for the instruction.

ENO	Condition	OUT
1	No errors detected	Valid characters
0	L is less than or equal to 0	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

RIGHT instruction

Parameter	Parameter type	Data type	Description
IN	IN	String	Input string
L	IN	Int	Length of the substring to be created, using the right-most L characters of the IN string
OUT	OUT	String	Output string

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.

The following table shows the condition codes for the instruction.

ENO	Condition	OUT
1	No errors detected	Valid characters
0	L is less than or equal to 0	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 instruction

Parameter	Parameter type	Data type	Description
IN	IN	String	Input string
L	IN	Int	Length of the substring to be created, using L characters of the IN string, beginning at character position P
P	IN	Int	Position of first substring character to be copied: P= 1, for the initial character position of the IN string
OUT	OUT	String	Output string

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. The following table shows the condition codes for the instruction.

ENO	Condition	OUT
1	No errors detected	Valid characters
0	L or P is less than or equal to 0	Current length is set to 0
	P is greater than maximum length of IN	
	Substring length (L) to be copied is larger than maximum length of OUT string	Characters are copied beginning at position P until the maximum length of OUT is reached

DELETE instruction

Parameter	Parameter 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

DELETE (Delete substring) 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.

The following table shows the condition codes for the instruction.

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
	L is less than 0, or P is less than or equal to 0	Current length is set to 0
	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

INSERT

Parameter	Parameter 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

INSERT (Insert substring) inserts string IN2 into string IN1. Insertion begins after the character at position P. The following table shows the condition codes for the instruction.

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 or equal to 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

REPLACE

Parameter	Parameter 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

REPLACE (Replace substring) 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.

- 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.

The following table shows the condition codes for the instruction.

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
	L is less than 0, or P is less than or equal to 0	Current length is set to 0
	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

FIND

Parameter	Parameter 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

FIND (Find substring) provides the character position of the substring or character 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. The following table shows the condition codes for the instruction.

ENO	Condition	OUT
1	No errors detected	Valid character position
0	IN2 is larger than IN1	Character position is set to 0

6.2.4 Program control instructions

6.2.4.1 Reset scan cycle watchdog instruction

RE_TRIGR
EN
ENO

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

You can set the value for maximum scan cycle time in the PLC device configuration for "Cycle time".

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 OB80 is included in the user program, the PLC executes OB 80 where you may add program logic to create a special reaction. If OB80 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 PLC to transition to STOP mode.

In STOP mode, your program execution stops while PLC system communications and system diagnostics continue.

6.2.4.2 Stop scan cycle instruction

STP (Stop PLC scan cycle) puts the PLC in Stop mode. When the PLC 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 48)

If EN = TRUE, then the PLC will enter STOP mode, program execution stops, and the ENO state is meaningless. Otherwise, EN = ENO = 0.

6.2.4.3 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.

GET_ERROR

GetError
EN
ENO
ERROR

GET_ERROR indicates that a program block execution error has occurred and fills a predefined error data structure with detailed error information.

Parameter	Data type	Description
ERROR	ErrorStruct	Error data structure: You can rename the structure, but not the members within the structure.

ErrorStruct data element	Data type	Description
ERROR_ID	Word	Error identifier
FLAGS	Byte	Always set to 0.
REACTION	Byte	Reaction to the error: • 0 = Ignore; nothing written (write error) • 1 = Substitute: a 0 was used for the input value (read error) • 2 = Skip the instruction
BLOCK_TYPE	Byte	Block type where error occurred: • 1 = OB • 2 = FC • 3 = FB
PAD_0	Byte	Internal fill byte for alignment purposes, will be 0
CODE_BLOCK_NUMBER	UInt	Block number where error occurred
ADDRESS	UDInt	Internal memory location of instruction which encountered error

ErrorStruct data element	Data type	Description
MODE	Byte	Internal mapping for how the remaining fields will be interpreted to be used by STEP 7 Basic
PAD_1	Byte	Internal fill byte for alignment purposes; not used, will be 0
OPERAND_NUMBER	UInt	Internal instruction operand number
POINTER_NUMBER_ LOCATION	UInt	(A) Internal instruction pointer location
SLOT_NUMBER_SCOPE	UInt	(B) Internal memory storage location
AREA	Byte	(C) Memory area referenced when the error was encountered: • L: 16#40 – 4E, 86, 87, 8E, 8F, C0 – CE • I: 16#81 • Q: 16#82 • M: 16#83 • DB: 16#84, 85, 8A, 8B
PAD_2	Byte	Internal fill byte for alignment purposes; not used, will be 0
DB_NUMBER	UInt	(D) DB which was referenced when a DB error occurred, 0 otherwise
OFFSET	UDInt	(E) The bit offset referenced when the error occurred (example: 12 = byte 1, bit 4)

GET_ERR_ID

GetErrorID
EN
ENO
ID

GET_ERR_ID indicates that a program block execution error has occurred and reports the ID (identifier code) of the error.

Parameter	Data type	Description
ID	Word	Error identifier values for the ErrorStruct ERROR_ID member

ERROR_ID Hexadecimal	ERROR_ID Decimal	Program block execution 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
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 Eeist
253C	9532	Wrong version or FC does not exist
253D	9533	Instruction does not exist

ERROR_ID Hexadecimal	ERROR_ID Decimal	Program block execution error
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 and transitioning to STOP mode. However, if you place one or more GET_ERROR or ERR_ID instructions within a code block, this block is now set to handle errors within the block. In this case, the CPU does not transition to STOP and does not log an error in the diagnostics buffer. Instead, the error information is reported in the output of the GET_ERROR or GET_ERR_ID instruction. You can read the detailed error information with the GET_ERROR instruction, or read just the error identifier with GET_ERR_ID instruction. Normally the first error is the most important, with the following errors only consequences of the first error.

The first execution of a GET_ERROR or GET_ERR_ID 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 GET_ERROR or GET_ERR_ID. Subsequent executions of either GET_ERROR or GET_ERR_ID return the first error since the previous execution of GET_ERROR or GET_ERR_ID. 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 GET_ERROR 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 ErrorStructs by using unique names. The members of an ErrorStruct cannot be renamed.

Error condition indicated by ENO

If EN = TRUE and GET_ERROR or GET_ERR_ID 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.

GET_ERROR and GET_ERR_ID 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.

6.2.5 Communications instructions

6.2.5.1 Open Ethernet Communication

Open Ethernet communication with automatic connect/disconnect (TSEND_C and TRCV_C)

Note

The processing of the TSEND_C and TRCV_C instructions can take an undetermined amount of time. To ensure that these instructions are processed in every scan cycle, always call them from within the main program cycle scan, such as from a program cycle OB or from a code block that is called from the program cycle scan. Do **not** call these instructions from a hardware interrupt OB, a time-delay interrupt OB, a cyclic interrupt OB, an error interrupt OB, or a startup OB.

For information transferring data with these instructions, see the section on data consistency (Page 86).

TSEND_C description

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. TSEND_C combines the functions of TCON, TDISCON and TSEND.

The minimum size of data that you can transmit with the TSEND_C instruction is a byte.

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 as the DATA parameter of the TRCV_C instruction.

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 description

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. The TRCV_C instruction combines the functions of the TCON, TDISCON, and TRCV instructions.

The minimum size of data that you can receive with the TRCV_C instruction is a byte. The TRCV_C instruction does not support the transmission of Boolean data or Boolean arrays.

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 as the DATA parameter of the TRCV_C instruction.

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.

Receive modes

TRCV_C handles the same receive modes as the TRCV instruction. The following table shows how data is entered in the receive area.

Protocol variant	Entering the data in the receive area	Parameter" connection_type"
TCP	Data reception with specified length	B#16#11
ISO on TCP	protocol-controlled	B#16#12

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 values 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.

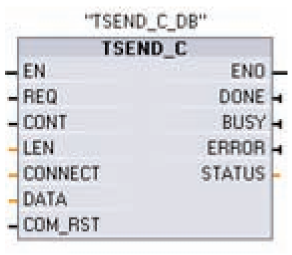
Due to the asynchronous processing of TRCV_C, the data in the receiver area are only consistent when parameter DONE = 1.

The following table shows the relationships between parameters BUSY, DONE and ERROR.

BUSY	DONE	ERROR	Description
TRUE	irrelevant	irrelevant	The job is being processed.
FALSE	TRUE	FALSE	The job successfully completed.

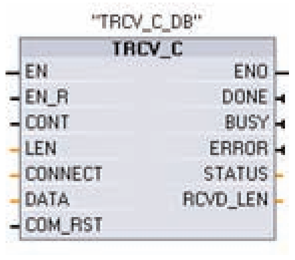
BUSY	DONE	ERROR	Description
FALSE	FALSE	TRUE	The job was ended with an error. The cause of the error can be found in the STATUS parameter.
FALSE	FALSE	FALSE	A new job was not assigned.

TSEND_C parameters



Parameter	Parameter type	Data type	Description
REQ	INPUT	Bool	Control parameter REQ starts the send job with the connection described in CONNECT on a rising edge.
CONT	INPUT	Bool	• 0: disconnect • 1: establish and hold connection
LEN	INPUT	Int	Maximum number of bytes to be sent. (Default = 0, which means that the DATA parameter determines the length of the data to be sent.).
CONNECT	IN_OUT	TCON-Param	Pointer to the connection description
DATA	IN_OUT	Variant	Send area; contains address and length of data to be sent.
COM_RST	IN_OUT	Bool	• 1: Complete restart of the function block, existing connection will be terminated.
DONE	OUTPUT	Bool	• 0: Job not yet started or still running. • 1: Job executed without error.
BUSY	OUTPUT	Bool	• 0: Job is completed. • 1: Job is not yet completed. A new job cannot be triggered.
ERROR	OUTPUT	Bool	• 1: Error occurred during processing. STATUS provides detailed information on the type of error.
STATUS	OUTPUT	Word	Error information

TRCV_C parameters



Parameter	Parameter type	Data type	Description
EN_R	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	Control parameter CONT: • 0: disconnect • 1: establish and hold connection
LEN	IN	Int	Length of the receive area in bytes. (Default = 0, which means that the DATA parameter determines the length of the data to be sent.).
CONNECT	IN_OUT	TCON-Param	Pointer to the connection description
DATA	IN_OUT	Variant	Receive area contains start address and maximum length of received data.
COM_RST	IN_OUT	Bool	• 1: Complete restart of the function block; existing connection will be terminated.
DONE	OUT	Bool	• 0: Job not yet started or still running. • 1: Job executed without error.
BUSY	OUT	Bool	• 0: Job is completed. • 1: Job is not yet completed. A new job cannot be triggered.
ERROR	OUT	Bool	• 1: Error occurred during processing. STATUS provides detailed information on the type of error.
STATUS	OUT	Word	Error information
RCVD_LEN	OUT	Int	Amount of data actually received, in bytes

Parameters Error and Status

ERROR	STATUS (W#16#...)	Description
0	0000	Job executed without error
0	7000	No job processing active
0	7001	Start job processing, establishing connection, waiting for connection partner

ERROR	STATUS (W#16#...)	Description
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 larger than the memory area specified in DATA; receiving memory area is too small
1	8089	The parameter 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	80A7	Communications error: you have called TDISCON before TCON was complete (TDISCON must first completely terminate the connection referenced by the ID)
1	80B2	The parameter 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
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	80C3	All connection resources are in use.

ERROR	STATUS (W#16#...)	Description
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 not possible (e.g. DB not available)
1	877F	CONNECT parameter: Internal error such as an invalid ANY reference

Open Ethernet communication with connect/disconnect control

Note

The processing of the TCON, TDISCON, TSEND, and TRCV instructions can take an undetermined amount of time. To ensure that these instructions are processed in every scan cycle, always call them from within the main program cycle scan, such as from a program cycle OB or from a code block that is called from the program cycle scan. Do **not** call these instructions from a hardware interrupt OB, a time-delay interrupt OB, a cyclic interrupt OB, an error interrupt OB, or a startup OB.

Ethernet communication using TCP and ISO on TCP protocols

These program instructions control the communication process:

- TCON makes a connection.
- TSEND and TRCV send and receive data.
- TDISCON breaks the connection.

The minimum size of data that you can transmit or receive with the TSEND and TRCV instructions is a byte. The TRCV instruction does not support the transmission of Boolean data or Boolean arrays. For information transferring data with these instructions, see the section on data consistency (Page 86).

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 as the DATA parameter of the TRCV instruction.

Both communication partners execute the TCON instruction to set up and establish the communications 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 reestablish 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 reestablish the connection, you must execute TCON again.

Functional description

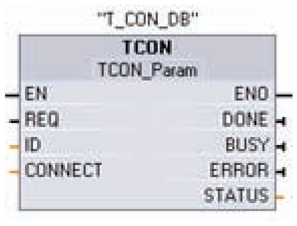
TCON, TDISCON, TSEND, and TRCV operate asynchronously, which means that the job processing extends over multiple instruction executions.

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.

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.

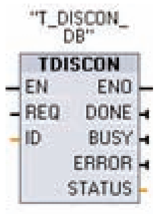
TCON



Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Control parameter REQUEST starts the job for establishing the connection specified by ID. The job starts at rising edge.
ID	IN	CONN_OUC (Word)	Reference to the connection to be established to the remote partner, or between the user program and the communication layer of the operating system. ID must be identical to the associated parameter ID in the local connection description. Value range: W#16#0001 to W#16#0FFF
CONNECT	IN_OUT	TCON-Param	Pointer to the connection description
DONE	OUT	Bool	Status parameter DONE: • 0: Job not yet started or still running • 1: Job executed without error

Parameter	Parameter type	Data type	Description
BUSY	OUT	Bool	BUSY = 1: Job is not yet complete BUSY = 0: Job is complete
ERROR	OUT	Bool	Status parameter ERROR: ERROR = 1: An error occurred in job processing. STATUS provides detailed information on the type of error.
STATUS	OUT	Word	Status parameter STATUS: Error information

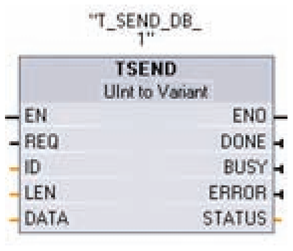
TDISCON



TCP and ISO on TCP: TDISCON terminates a communications connection from the CPU to a communication partner.

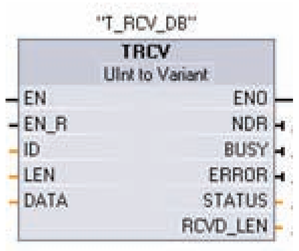
Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Control parameter REQUEST starts the job for establishing the connection specified by ID. The job starts at rising edge.
ID	IN	CONN_OUC (Word)	Reference to the connection to be terminated to the remote partner or between the user program and the communications level of the operating system. ID must be identical to the associated parameter ID in the local connection description. Value range: W#16#0001 to W#16#0FFF
DONE	OUT	Bool	Status parameter DONE: • 0: Job not yet started or still running • 1: Job executed without error
BUSY	OUT	Bool	BUSY = 1: Job is not yet complete BUSY = 0: Job is complete
ERROR	OUT	Bool	ERROR = 1: Error occurred during processing.
STATUS	OUT	Word	Error code

TSEND



Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Control parameter REQUEST starts the send job on a rising edge. The data is transferred from the area specified by DATA and LEN.
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	Int	Maximum number of bytes to be sent with the job
DATA	IN_OUT	Variant	Pointer to data area to send: Sender area; contains address and length. The address refers to: • The process image input table • The process image output table • A bit memory • A data block
DONE	OUT	Bool	Status parameter DONE: • 0: Job not yet started or still running. • 1: Job executed without error.
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	Status parameter ERROR: ERROR = 1: Error occurred during processing. STATUS provides detailed information on the type of error
STATUS	OUT	Word	Status parameter STATUS: Error information

TRCV



Parameter	Parameter type	Data type	Description
EN_R	IN	Bool	Control parameter enabled to receive: With EN_R = 1, TRCV is ready to receive. The receive job is being 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	Int	Length of the receive area in bytes (Default = 0, which means that the DATA parameter determines the length of the data to be received.).
DATA	IN_OUT	Variant	Pointer to received data: Receive area that contains address and length. The address refers to: • The process image input table • The process image output table • A bit memory • A data block
NDR	OUT	Bool	Status parameter NDR: • 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	Error information
RCVD_LEN	OUT	Int	Amount of data actually received, in bytes

Receive area

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 as the DATA parameter of the TRCV instruction.

The following table shows how TRCV enters the received data in the receive area.

Protocol variant	Entering the data in the receive area	Parameter connection type
TCP	Data reception with specified length	B#16#11
ISO on TCP	Protocol-controlled	B#16#12

As soon as all the job data has been received, TRCV transfers it to the receive area and sets NDR to 1.

Condition codes for TCON

ERROR	STATUS (W#16#...)	Explanation
0	0000	Connection was established successfully
0	7000	No job processing active
0	7001	Start job processing, establishing connection
0	7002	Follow-on call (REQ irrelevant), connection being established
1	8086	The ID parameter is outside the permitted range.
1	8087	Maximum number of connections reached; no additional connection possible
1	809B	The local_device_id in the connection description does not match the CPU.
1	80A1	Connection or port is already occupied by user
1	80A2	Local or remote port is occupied by the system
1	80A3	Attempt being made to re-establish an existing connection
1	80A4	IP address of the remote connection end point is invalid; it may match the local IP address
1	80A7	Communications error: you executed TDISCON before TCON was complete. TDISCON must first completely terminate the connection referenced by the ID.
1	80B3	Inconsistent parameter assignment: Group error for the error codes W#16#80A0 to W#16#80A2, W#16#80A4, W#16#80B4 to W#16#80B9

ERROR	STATUS (W#16#...)	Explanation
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	80B5	Error in parameter active_est
1	80B6	Parameter assignment error in parameter connection_type
1	80B7	Error in one of the following parameters: block_length, local_tsap_id_len, rem_subnet_id_len, rem_staddr_len, rem_tsap_id_len, next_staddr_len
1	80B8	Parameter in the local connection description and Parameter ID are different
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 TDISCON.

Condition codes for TDISCON

ERROR	STATUS (W#16#...)	Explanation
0	0000	Connection was terminated successfully
0	7000	No job processing active
0	7001	Start of job processing, connection being terminated
0	7002	Follow-on call (REQ irrelevant), connection being terminated
1	8086	The ID parameter is not in the permitted address range.
1	80A3	Attempt being made to terminate a non-existent connection
1	80C4	Temporary communications error: The interface is receiving new parameters or the connection is currently being established.

Condition codes for TSEND

ERROR	STATUS (W#16#...)	Explanation
0	0000	Send job completed without error
0	7000	No job processing active
0	7001	Start of job processing, data being sent: During this processing the operating system accesses the data in the DATA send area.

ERROR	STATUS (W#16#...)	Explanation
0	7002	Follow-on call (REQ irrelevant), job being processed: The operating system accesses the data in the DATA send area during this processing.
1	8085	LEN parameter is greater than the largest permitted value.
1	8086	The ID parameter is not in the permitted address range
1	8088	LEN parameter is larger than the memory area specified in DATA
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	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 parameters or the connection is currently being established.

Condition codes for TRCV

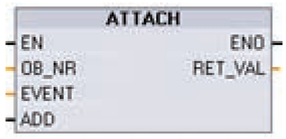
ERROR	STATUS (W#16#...)	Explanation
0	0000	New data accepted: The current length of the received data is shown in RCVD_LEN.
0	7000	Block not ready to receive
0	7001	Block is ready to receive, receive job was activated.
0	7002	Follow-on call, receive job being processed: Data is written to the receive area during this processing For this reason, an error could result in inconsistent data in the receive area.
1	8085	The LEN parameter is greater than the largest permitted value, or you changed the LEN or DATA parameter since the first call.
1	8086	The ID parameter is not in the permitted address range
1	8088	Receive area is too small: The ·Value LEN is greater than the receive area specified by DATA.
1	80A1	Communications error: • The specified connection has not yet been established • The specified connection is currently being terminated. A receive job over this connection is not possible. • The interface is receiving new parameters.
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 partner cannot be established at the moment. • The interface is receiving new parameter settings or the connection is currently being established.

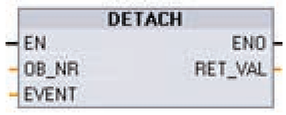
6.2.5.2 Point-to-Point instructions

The Point-to-Point (PtP) chapter (Page 237) provides detailed information about the PtP instructions and the communication modules.

6.2.6 Interrupt instructions

6.2.6.1 Attach and detach instructions





You can activate and deactivate interrupt event-driven subprograms with the ATTACH and DETACH instructions.

- ATTACH enables interrupt OB subprogram execution for a hardware interrupt event.
- DETACH disables interrupt OB subprogram execution for a hardware interrupt event.

Parameter	Parameter type	Data type	Description
OB_NR	IN	Int	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	DWord	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 plus any signal board 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 plus any signal board input)
  - 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

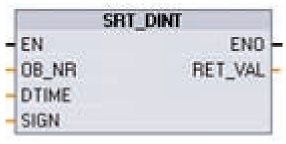
Use the DETACH instruction to detach either a particular event or all events from a particular OB. If an EVENT is specified, then only this one event is detached from the specified OB_NR; any other events currently attached to this OB_NR will remain attached. If no EVENT is specified, then all events currently attached to OB_NR will be detached.

Condition codes

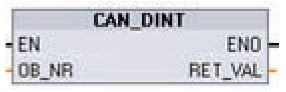
RET_VAL (W#16#....)	ENO status	Description
0000	1	No error
0001	0	Nothing to Detach (DETACH only)
8090	0	OB does not exist
8091	0	OB is wrong type
8093	0	Event does not exist

6.2.6.2 Start and cancel time delay interrupt instructions

You can start and cancel time delay interrupt processing with the SRT_DINT and CAN_DINT instructions. 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.



SRT_DINT starts a time delay interrupt that executes an OB (organization block) subprogram when the delay time specified by parameter DTIME has elapsed.



CAN_DINT cancels a time delay interrupt that has already started. The time delay interrupt OB is not executed in this case.

SRT_DINT parameters

Parameter	Parameter type	Data type	Description
OB_NR	IN	Int	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	IN	Time	Time delay value (1 to 60000 ms) You can create longer delay times, for example, by using a counter inside a time delay interrupt OB.
SIGN	IN	Word	Not used by the S7-1200; any value is accepted
RET_VAL	OUT	Int	Execution condition code

CAN_DINT parameters

Parameter	Parameter type	Data type	Description
OB_NR	IN	Int	Time delay interrupt OB identifier. You can use an OB number or symbolic name.
RET_VAL	OUT	Int	Execution condition code

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 time 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:

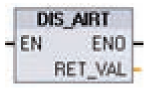
1. Double-click the "Add new block" item in the "Program blocks" branch of the project tree, select "Organization block (OB)", and choose "Time delay interrupt".
2. You have the option to rename the OB, select the programming language, or select the block number. Switch to manual numbering if you want 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.

Condition codes

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

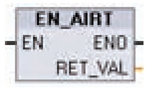
6.2.6.3 Disable and Enable alarm interrupt instructions

Use the DIS_AIRT and EN_AIRT instructions to disable and enable alarm interrupt processing.



DIS_AIRT delays the processing of new interrupt events. You can execute DIS_AIRT more than once in an OB. 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.

Once they 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.



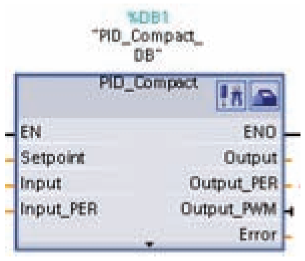
EN_AIRT enables the processing of interrupt events that you previously disabled with the DIS_AIRT instruction. Each DIS_AIRT execution must be cancelled by an EN_AIRT execution. If, for example, you have disabled interrupts five times with five DIS_AIRT executions, you must cancel these with five EN_AIRT executions.

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.

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.

Parameter	Parameter type	Data type	Description
RET_VAL	OUT	Int	Number of delays = number of DIS_AIRT executions in the queue.

6.2.7 PID control



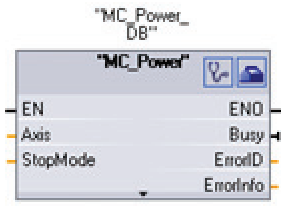
The "PID_Compact" statement makes a PID controller with optimizing self tuning for automatic and manual mode available.

For information about the PID_Compact instruction, refer to the online help of the TIA portal.

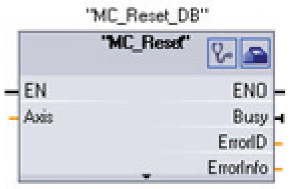
6.2.8 Motion control instructions

The motion control instructions use an associated technology data block and the dedicated PTO (pulse train outputs) of the CPU to control the motion on an axis. For information about the motion control instructions, refer to the online help of STEP 7 Basic.

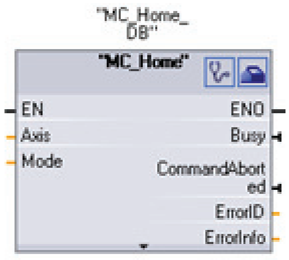
NOTICE
The maximum pulse frequency of the pulse output generators is 100 KHz for the digital outputs of the CPU and 20 KHz for the digital outputs of the signal board. However, STEP 7 Basic does not alert you when you configure an axis that with a maximum speed or frequency that exceeds this hardware limitation. This could cause problems with your application, so always ensure that you do not exceed the maximum pulse frequency of the hardware.



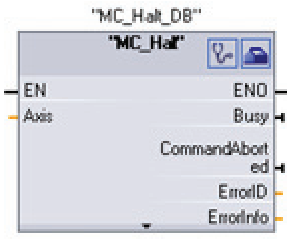
MC_Power enables and disables a motion control axis.



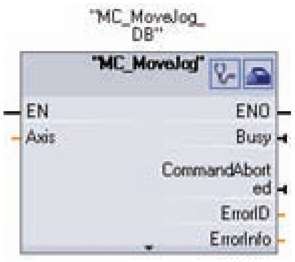
MC_Reset resets all motion control errors. All motion control errors that can be acknowledged are acknowledged.



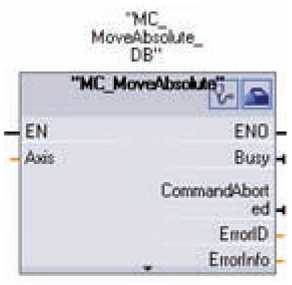
MC_Home establishes the relationship between the axis control program and the axis mechanical positioning system.



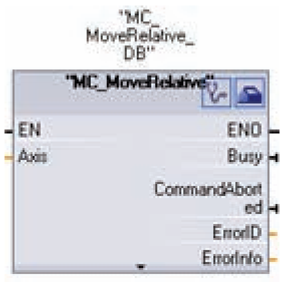
MC_Halt cancels all motion processes and causes the axis motion to stop. The stop position is not defined.



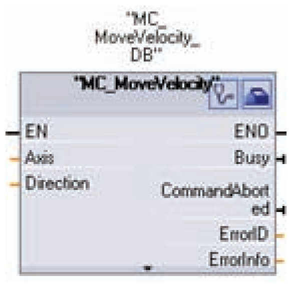
MC_MoveJog executes jog mode for testing and startup purposes.



MC_MoveAbsolute starts motion to an absolute position. The job ends when the target position is reached.



MC_MoveRelative starts a positioning motion relative to the start position.



MC_MoveVelocity causes the axis to travel with the specified speed.

Note

Pulse-train outputs cannot be used by other instructions in the user program

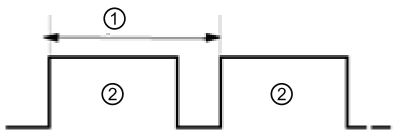
When you configure the outputs of the CPU or signal board as pulse generators (for use with the PWM or basic motion control instructions), the corresponding outputs addresses (Q0.0, Q0.1, Q4.0, and Q4.1) are removed from the Q memory and cannot be used for other purposes in your user program. If your user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.

6.2.9 Pulse instruction

6.2.9.1 CTRL_PWM instruction

The CTRL_PWM Pulse Width Modulation (PWM) instruction 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 effect the desired control.



①	Cycle time
②	Pulse width

Pulse width can be expressed as hundreths of the cycle time (0 – 100), as thousandths (0 – 1000), as ten thousandths (0 – 10000), or as S7 analog format. The pulse width can vary from 0 (no pulse, always off) to full scale (no pulse, always on).

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.

Description	Default output assignment		
		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	--

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

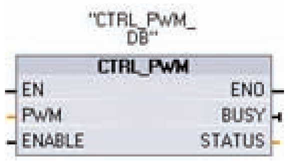
The maximum pulse frequency of the pulse output generators is 100 KHz for the digital outputs of the CPU and 20 KHz for the digital outputs of the signal board. However, STEP 7 Basic does not alert you when you configure an axis that with a maximum speed or frequency that exceeds this hardware limitation. 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 Signal Board
- Time base: milliseconds or microseconds

- Pulse width format:
  - Hundreths (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.

Output addresses



Start address: Enter the Q word address where you want to locate the pulse width value. 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.

Parameter	Parameter type	Data type	Initial value	Description
PWM	IN	Word	0	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.
ENABLE	IN	Bool		1=start pulse generator 0 = stop pulse generator
BUSY	OUT	Bool	0	Function busy
STATUS	OUT	Word	0	Execution condition code

Operation

A data block (DB) is used by the CTRL_PWM instruction to store parameter information. When placing a CTRL_PWM instruction into the program editor, a DB will be assigned. 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 S7-1200 processes the request when the CTRL_PWM instruction is executed, parameter BUSY will always report FALSE on S7-1200 CPU models.

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 PLC first enters the 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).

Condition codes

STATUS value	Description
0	No error
80A1	PWM identifier does not address a valid PWM

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.

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.

6.3 Global library instructions

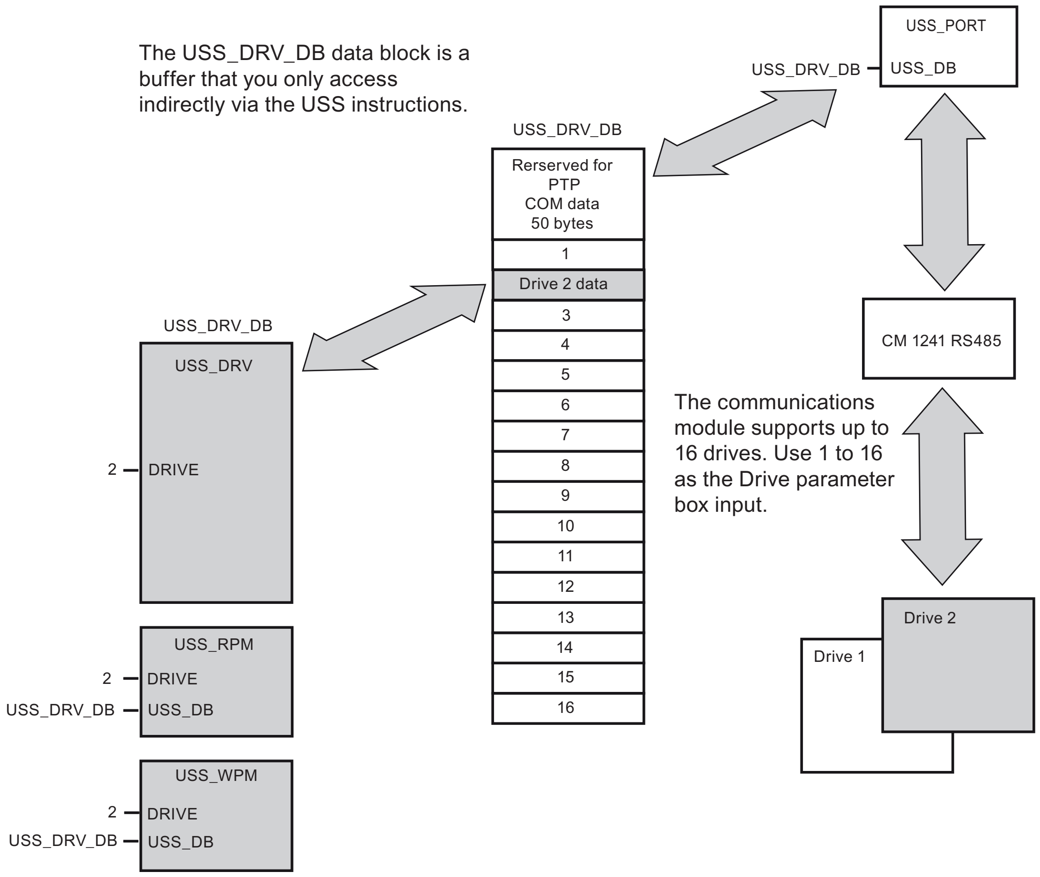
6.3.1 USS

The USS Protocol library makes controlling Siemens drives which support USS protocol. The instructions include functions that are specifically designed for using the USS protocol to communicate with the drive. The CM 1241 RS485 module communicates with the drives on RS485 ports. You can control the physical drive and the read/write drive parameters with the USS library.

6.3.1.1 Requirements for using the USS protocol

The library provides 1 FB and 3 FCs to support the USS protocol. Each CM 1241 RS485 communications module supports a maximum of 16 drives.

A single Instance Data Block contains temporary storage and buffers for all drives on the USS network connected to each PtP communication module you install. The USS functions for these drives share the information in this data block.


The USS_DRV_DB data block is a buffer that you only access indirectly via the USS instructions.
USS_PORT
USS_DRV_DB
USS_DB
USS_DRV_DB
Rerserved for PTP COM data 50 bytes
1
Drive 2 data
3
4
5
6
7
8
9
10
11
12
13
14
15
16
USS_DRV_DB
USS_DRV
2
DRIVE
CM 1241 RS485
The communications module supports up to 16 drives. Use 1 to 16 as the Drive parameter box input.
USS_RPM
2
DRIVE
USS_DRV_DB
USS_DB
USS_WPM
2
DRIVE
USS_DRV_DB
USS_DB
Drive 2
Drive 1

All drives (up to 16) connected to a single CM 1241 RS485 are part of the same USS network. All drives connected to a different CM 1241 RS485 are part of a different USS network. Because the S7-1200 supports up to three CM 1241 RS485 devices, you can have up to three USS networks, with up to 16 drives on each network, for a total of 48 USS drives supported.

Each USS network is managed using a unique data block (three data blocks are required for three USS networks using three CM 1241 RS485 devices). 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 editor, you will be prompted by the "Call options" dialog for which DB to assign 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 will be 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 appropriate DB that was previously assigned for this USS network.

Instructions USS_PORT, USS_RPM, and USS_WPM are all Functions (FC's). No DB is assigned when you place these FC's in the editor. Instead, you must assign the appropriate DB to the "USS_DB" input of these instructions (double-click on the parameter field, then click on the helper icon to see the available DB's).

The USS_PORT function handles actual communication between the CPU and the drives via the PtP communication module. 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 the Main 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 functions of the drive.

You may only call the USS_DRV function block from the main OB.

CAUTION
Only call USS_DRV, USS_RPM, USS_WPM from the Main OB. The USS_PORT function can be called from any OB, usually from a Time delay interrupt. Failure to prevent interruption of USS_PORT 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 OB.

Calculating the time required for communicating with the drive

Communications with the drive are asynchronous to the S7-1200 scan. 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 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.

Baud rate	Calculated minimum USS_PORT call Interval (milliseconds)	Drive message interval timeout per drive (milliseconds)
1200	790	2370
2400	405	1215
4800	212.5	638
9600	116.3	349
19200	68.2	205
38400	44.1	133
57600	36.1	109
115200	28.1	85

6.3.1.2 USS_DRV instruction

The USS_DRV instruction exchanges data with the 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 module must use the same Instance Data Block. You must create the DB name when you place the first USS_DRV instruction and you reuse that DB that was created by the initial instruction usage.

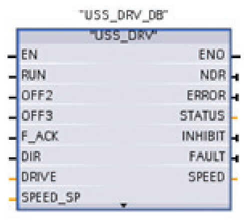
When the initial USS_DRV execution is made, the drive indicated by the USS address (parameter DRIVE) is initialized in the Instance DB. After this initialization, subsequent executions of USS_PORT can then begin communication to the drive at this drive number.

Changing the Drive number requires a PLC 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 are communicated with 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.

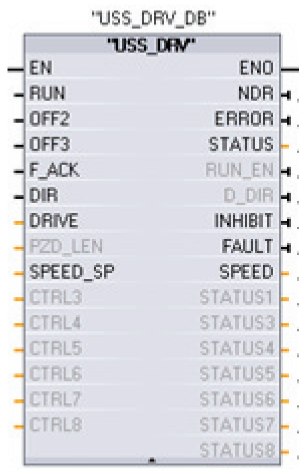
You can control the drive direction of rotation using either the DIR input (BOOL) or using the sign (positive or negative) with the SPEED_SP input (REAL). The following table indicates how these inputs work together to determine the drive direction, assuming the motor is wired for forward rotation.

SPEED_SP	DIR	Drive Direction
Value > 0	0	Reverse
Value > 0	1	Forward
Value < 0	0	Forward
Value < 0	1	Reverse

LAD (default view)



LAD (expanded view)



Expand the box to reveal all the parameters by clicking the bottom of the box.

The parameter pins that are grayed are optional and do not need to be assigned.

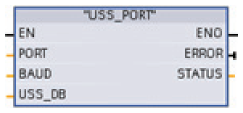
Parameter	Parameter type	Data type	Description
RUN	IN	Bool	Drive start bit: When true, this input enables the drive to run at the preset speed.
OFF2	IN	Bool	Electrical stop bit: When false, this bit cause the drive to coast to a stop with no braking.
OFF3	IN	Bool	Fast stop bit – When false, this bit causes a fast stop by causing 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. This 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. Default 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).
CTRL3	IN	UInt	Control word 3 – A value written to a user-configurable parameter on the drive. The user must configure this on the drive. Optional parameter.
CTRL4	IN	UInt	Control word 4 – A value written to a user-configurable parameter on the drive. The user must configure this on the drive. Optional parameter.
CTRL5	IN	UInt	Control word 5 – A value written to a user-configurable parameter on the drive. The user must configure this on the drive. Optional parameter.
CTRL6	IN	UInt	Control word 6 – A value written to a user-configurable parameter on the drive. The user must configure this on the drive.
CTRL7	IN	UInt	Control word 7 – A value written to a user-configurable parameter on the drive. The user must configure this on the drive. Optional parameter.
CTRL8	IN	UInt	Control word 8 – A value written to a user-configurable parameter on the drive. The user 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	UInt	The status value of the request. It indicates the result of the scan. This is not a status word returned from the drive.

Parameter	Parameter type	Data type	Description
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. The user must fix the problem and then set the F_ACK bit to clear this bit when set.
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	UInt	Drive Status Word 1 – This value contains fixed status bits of a drive.
STATUS3	OUT	UInt	Drive Status Word 3 – This value contains a user-configurable status word on the drive.
STATUS4	OUT	UInt	Drive Status Word 4 – This value contains a user-configurable status word on the drive.
STATUS5	OUT	UInt	Drive Status Word 5 – This value contains a user-configurable status word on the drive.
STATUS6	OUT	UInt	Drive Status Word 6 – This value contains a user-configurable status word on the drive.
STATUS7	OUT	UInt	Drive Status Word 7 – This value contains a user-configurable status word on the drive.
STATUS8	OUT	UInt	Drive Status Word 8 – This value contains a user-configurable status word on the drive.

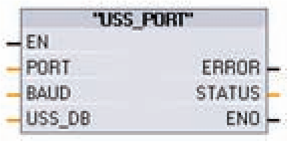
6.3.1.3 USS_PORT instruction

The USS_PORT instruction handles communication over the USS network. Typically there is only one USS_PORT function per PtP communication module in the program, and each call of this function handles a transmission to or from a single drive. Your program must execute the USS_PORT function often enough to prevent drive timeouts. All USS functions associated with one USS network and PtP communication module must use the same Instance Data Block. USS_PORT is usually called from a time delay interrupt OB to prevent drive timeouts and keep the most recent USS data updates available for USS_DRV calls.

LAD



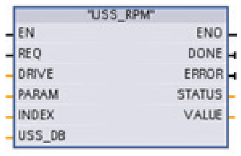
FBD



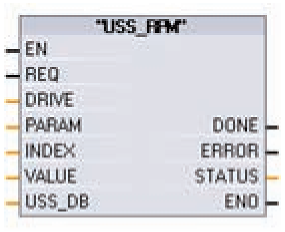
Parameter	Parameter type	Data type	Description
PORT	IN	Port	PtP communications module. Identifier: This a constant which can be referenced within the "Constants" tab of the default tag table.
BAUD	IN	DInt	The Baud Rate to be used for USS communication.
USS_DB	IN	DInt	This is a reference to the instance DB that is created and initialized when a USS_DRV instruction is placed in your program.
ERROR	OUT	Bool	When true, this pin indicates that an error has occurred and the STATUS output is valid.
STATUS	OUT	UInt	The status value of the request. It indicates the result of the scan or initialization. Additional information is available in the "USS_Extended_Error" variable for some status codes.

6.3.1.4 USS_RPM instruction

LAD



FBD

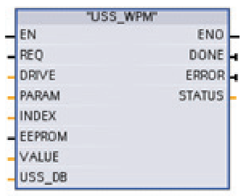
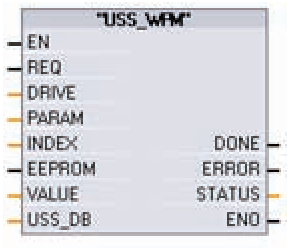


The USS_RPM instruction reads a parameter from the drive. All USS functions associated with one USS network and PtP communication module must use the same data block. USS_RPM must be called from the main OB.

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Send request: When true, it 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: This input is the address of the USS drive. The valid range is drive 1 to drive 16.
PARAM	IN	UInt	Parameter number: This input designates which drive parameter is written. The range of this parameter is 0 to 2047. See your drive manual for details on how to access any parameters above this range.
INDEX	IN	UInt	Parameter index: This input 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	IN	Variant	This is a reference to the instance DB that is created and initialized when a USS_DRV instruction is placed in your program.

Parameter	Parameter type	Data type	Description
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	OUT	Bool	Done: 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
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	UInt	This is the status value of the request. It indicates the result of the read request. Additional information is available in the "USS_Extended_Error" variable for some status codes.

6.3.1.5 USS_WPM instruction

LAD	FBD	
 "USS_WPM" EN REQ DRIVE PARAM INDEX EEPROM VALUE USS_DB ENO DONE ERROR STATUS	 "USS_WPM" EN REQ DRIVE PARAM INDEX EEPROM VALUE USS_DB DONE ERROR STATUS ENO	The USS_WPM instruction modifies a parameter in the drive. All USS functions associated with one USS network and PtP communication module must use the same data block. USS_WPM must be called from the main OB.

Note

EEPROM write operations

Beware of overusing the EEPROM permanent write operation. Minimize the number of EEPROM write operations to extend the EEPROM life.

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Send request: When true, it 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: This input is the address of the USS drive. The valid range is drive 1 to drive 16.
PARAM	IN	UInt	Parameter number: This input designates which drive parameter is written. The range of this parameter is 0 to 2047. See your drive manual for details on how to access any parameters above this range.
INDEX	IN	UInt	Parameter index: This input 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, writes to the drive parameter will be stored in the drive EEPROM. If false, the write is temporary and will not be retained if 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	IN	Variant	This is a reference to the instance DB that is created and initialized when a USS_DRV instruction is placed in your program.
DONE	OUT	Bool	Done: When TRUE 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 drive's confirmation that the write is complete via another USS_WPM poll or on the second of the next two calls to USS_DRV.
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	UInt	This is the status value of the request. It indicates the result of the write request. Additional information is available in the "USS_Extended_Error" variable for some status codes.

6.3.1.6 USS status codes

USS instruction status codes are returned at the STATUS output of the USS functions.

STATUS value (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
8182	User supplied a Word for a parameter value and received a DWord or Real from the drive in the response
8183	User supplied a DWord or Real for a parameter value and received a Word from the drive in the response
8184	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: 1-16)
8186	Speed set point out of valid range (valid speed SP range: -200% to 200%)
8187	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	Parameter request channel is in use by another request for this drive
818B	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	Drive returned an extended error on a parameter request operation. See the extended error description below this table.
818D	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	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 of 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 (Point-to-Point) 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.

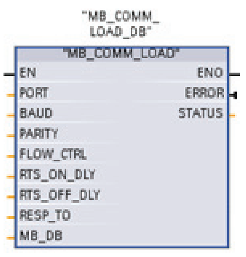
USS drive extended error codes

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 like 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 of the USS_DRV Instance DB. 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 the "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.

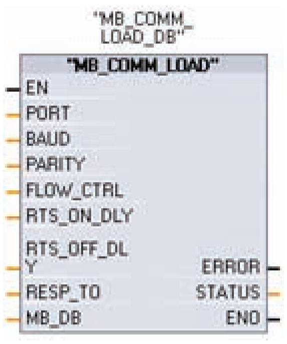
6.3.2 MODBUS

6.3.2.1 MB_COMM_LOAD

LAD



FBD



The MB_COMM_LOAD instruction configures a port on the Point-to-Point (PtP) CM 1241 RS485 or CM 1241 RS232 module for Modbus RTU protocol communications.

Parameter	Parameter type	Data type	Description
PORT	IN	UInt	Communications port identifier: After you install the CM module in the Device configuration, the port identifier appears in the helper drop-list available at the PORT box connection. This constant can also be referenced within the "Constants" tab of the default 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
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 RS-485 ports). RTS delays shall be applied independent of the FLOW_CTRL selection.

Parameter	Parameter type	Data type	Description
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 RS-485 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, if the specified number of retries has been sent. 5 ms to 65535 ms (default value = 1000ms).
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 helper drop-list available at the MB_DB box connection.
ERROR	OUT	Bool	Error: • 0 – No error detected • 1 – Indicates that an error was detected and the error code at parameter STATUS is valid
STATUS	OUT	Word	Port configuration error code

MB_COMM_LOAD is executed to configure a port for the Modbus RTU protocol. After the port is configured, you communicate on the Modbus by executing either MB_SLAVE or MB_MASTER instructions.

MB_COMM_LOAD should be called one time to initialize the port. MB_COMM_LOAD only needs to be called again if one of the communication parameters must change. You can call MB_COMM_LOAD from a startup OB and execute it one time, or use the first scan system flag to initiate the call to execute it one time.

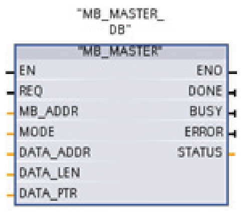
One instance of the MB_COMM_LOAD must be used to configure each port of each communication module that is used for Modbus communication. You must assign a unique MB_COMM_LOAD Instance Data Block for each port that you use. The S7-1200 CPU is limited to 3 communication modules.

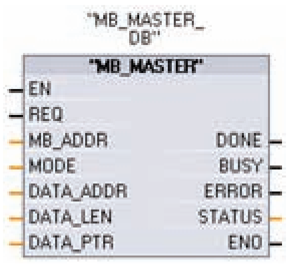
An Instance Data Block is assigned when you place the MB_MASTER or MB_SLAVE instructions. This Instance Data Block is referenced when you specify the MB_DB parameter on the MB_COMM_LOAD instruction.

STATUS value (W#16#....)	Description
0000	No error
8180	Invalid port ID value
8181	Invalid baud rate value
8182	Invalid parity value
8183	Invalid flow control value
8184	Invalid response timeout value
8185	Incorrect MB_DB pointer to the instance DB for an MB_MASTER or MB_SLAVE

6.3.2.2 MB_MASTER

LAD FBD





The MB_MASTER instruction allows your program to communicate as a Modbus master using a port on the Point-to-Point (PtP) CM 1241 RS485 or CM 1241 RS232 module. You can access data in one or more Modbus slave devices.

An Instance Data Block is assigned when you place the MB_MASTER instruction in your program. This MB_MASTER Instance Data Block name is used when you specify the MB_DB parameter on the MB_COMM_LOAD instruction.

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Request Input: • 0 – No request • 1 – Request to transmit data to Modbus Slave(s)
MB_ADR	IN	USInt	Modbus RTU station address: Valid address range: 0 to 247 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 following Modbus functions table 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 CPU DB address for the data being written or read. The DB must be a "NOT symbolic access only" DB type. See the DATA_PTR note below.
NDR	OUT	Bool	New Data Ready: • 0 – Transaction not complete • 1 – Indicates that the MB_MASTER instruction has completed the requested transaction with Modbus slave(s)
BUSY	OUT	Bool	Busy: • 0 – No MB_MASTER transaction in progress • 1 – MB_MASTER transaction in progress
ERROR	OUT	Bool	Error: • 0 – No error detected • 1 – Indicates that an error was detected and the error code supplied at parameter STATUS is valid
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 cannot be used by MB_SLAVE . One or more instances of MB_MASTER execution can be used with 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.
- If your program operates a Modbus master and uses MB_MASTER to send a request to a slave, then you must continue to poll (execute MB_MASTER) until the response from the slave is returned.
- Call all MB_MASTER execution for a given port from the same OB (or OB priority level).

REQ parameter

REQ value FALSE = No request

REQ value TRUE = Request to transmit data to Modbus Slave(s).

You must supply this input through an positive edge triggered contact on the first call for MB_MASTER execution. The edge triggered pulse will invoke the transmission request once. All inputs are captured and held unchanged for one request and response triggered by this input.

Internally the MB_MASTER will start a state machine to make sure that no other MB_MASTER instruction is allowed to issue a request until this request has been completed.

In addition, if the same instance of the call to the MB_MASTER FB is executed again with the REQ input TRUE before the completion of the request, then no subsequent transmissions will be made. However, as soon as the request has been completed, a new request will be issued if MB_MASTER is executed with the REQ input set to true.

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.

MB_MASTER uses a MODE input rather than a Function Code input. The combination of MODE and Modbus address range determine the Function Code that is used in the actual Modbus message. The following table shows the correspondence between MBUS_MASTER parameter MODE, Modbus function code, and Modbus address range.

MB_MASTER Modbus functions				
	Modbus address parameter DATA_ADDR	Address type	Modbus data length parameter DATA_LEN	Modbus function
Mode 0				
Read	00001 to 09999	Output bits	1 to 2000	01H
	10001 – 19999	Input bits	1 to 2000	02H
	30001 - 39999	Input registers	1 to 125	04H
	40001 to 49999 400001 to 465536(Extended)	Holding registers	1 to 125	03H
Mode 1				
Write	00001 to 09999	Output bits	1 (single bit)	05H
	40001 to 49999 400001 to 465536(Extended)	Holding registers	1 (single word)	06H
	00001 to 09999	Output bits	2 to 1968	15H
	40001 to 49999 400001 to 465536(Extended)	Holding registers	2 to 123	16H
Mode 2				
Some Modbus slaves do not support single bit or word writes with Modbus functions 05H and 06H. In these cases, Mode 2 is used to force single bit and word writes using Modbus functions 15H and 16H.				
Write	00001 to 09999	Output bits	1 to 1968	15H
	40001 to 49999 400001 to 465536(Extended	Holding registers	1 to 123	16H
Mode 11 • Reads an event counter word from the Modbus slave that is referenced as an input to MB_ADDR • On a Siemens S7-1200 Modbus slave, this counter is incremented every time that the slave receives a valid read or write (non-broadcast) request from a Modbus master. • The returned value is stored in the word location specified as the input to DATA_PTR. • A valid DATA_LEN is not required for this mode.				
Mode 80 • Checks the communication status of the Modbus slave that is referenced as an input to MB_ADDR • The setting of the NDR output bit on the MB_MASTER instruction indicates that the addressed Modbus slave responded with the appropriate response data. • No data is returned to your program. • A valid DATA_LEN is not required for this mode.				
Mode 81 • Resets the event counter (as returned by Mode 11) on the Modbus slave that is referenced as an input to MB_ADDR • The setting of the NDR output bit on the MB_MASTER instruction indicates that the addressed Modbus slave responded with the appropriate response data. • No data is returned to your program. • A valid DATA_LEN is not required for this mode.				

DATA_PTR parameter

The DATA_PTR parameter points to the local source or destination address (the address in the S7-1200 CPU) of the data that is written to or read from, respectively. When you use the MB_MASTER instruction to create a Modbus master, you must create a global data block that provides data storage for reads and writes to Modbus slaves.

Note

The DATA_PTR parameter must reference a global data block which was created with the Symbolic Access Only attribute box unchecked.

You must uncheck the "Symbolic address only" box when you add a new Data block to create a classic global DB type.

Data block structures for the DATA_PTR parameter

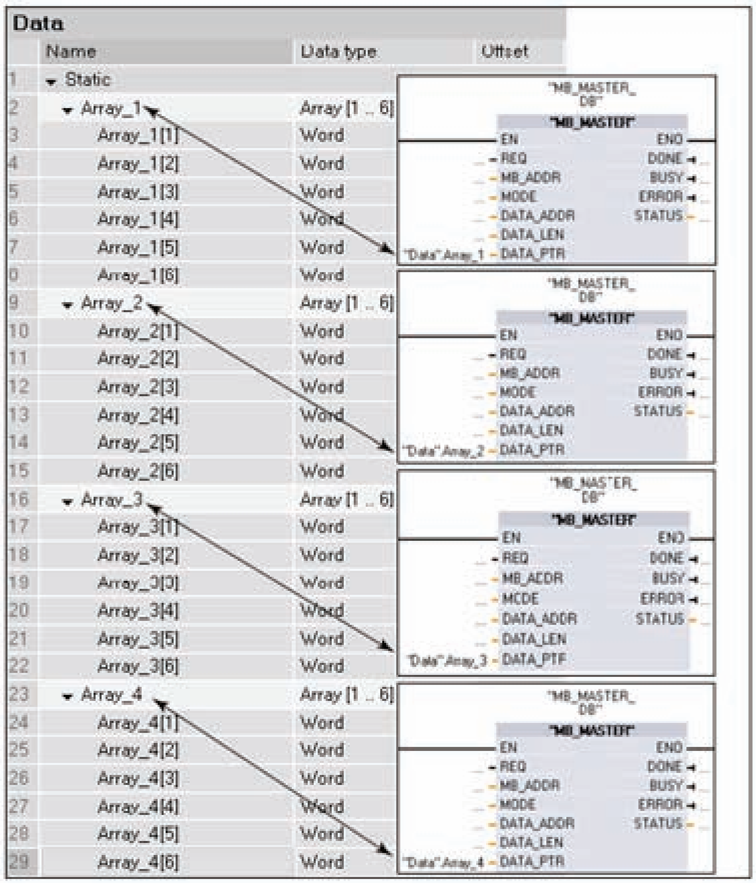
- These data types are valid for **word reads** of Modbus addresses 30001 to 39999, 40001 to 49999, and 400001 to 465536 and also for **word writes** to Modbus addresses 40001 to 49999 and 400001 to 465536.
  - Standard array of WORD, UINT, or INT data types, as shown below.
  - Named WORD, UINT, or 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 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 area in a global data block. 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 area of a global data block.
- 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.
- All of the arrays in the example below are created as base 1 arrays [1 … ##]. The arrays could have been created as base 0 arrays [0 … ###] or a mix of base 0 and base 1.

Example MB_MASTER instructions accessing DATA_PTR global data blocks

The example global data block shown below uses 4 uniquely named, 6 word arrays for Modbus request data storage. Although the data arrays in this example are the same size, the arrays can be of any size and are shown as the same size to simplify the examples. Each array could also be replaced with a data structure that includes more descriptive tag names and mixed data types. Examples of alternative data structures are provided in the HR_DB parameter description of the MB_SLAVE instruction (Page 199).

The MB_MASTER instruction examples below show only the DATA_PTR parameter and not the other required parameters. The purpose of the examples is to show how the MB_MASTER instruction uses the DATA_PTR data block.

The arrows indicate how each array is associated to different MB_MASTER instructions.



Data

	Name	Data type	Offset
1	Static		
2	Array_1	Array [1 .. 6]	
3	Array_1[1]	Word	
4	Array_1[2]	Word	
5	Array_1[3]	Word	
6	Array_1[4]	Word	
7	Array_1[5]	Word	
8	Array_1[6]	Word	
9	Array_2	Array [1 .. 6]	
10	Array_2[1]	Word	
11	Array_2[2]	Word	
12	Array_2[3]	Word	
13	Array_2[4]	Word	
14	Array_2[5]	Word	
15	Array_2[6]	Word	
16	Array_3	Array [1 .. 6]	
17	Array_3[1]	Word	
18	Array_3[2]	Word	
19	Array_3[3]	Word	
20	Array_3[4]	Word	
21	Array_3[5]	Word	
22	Array_3[6]	Word	
23	Array_4	Array [1 .. 6]	
24	Array_4[1]	Word	
25	Array_4[2]	Word	
26	Array_4[3]	Word	
27	Array_4[4]	Word	
28	Array_4[5]	Word	
29	Array_4[6]	Word	

The first element of any array or structure is always the first source or destination of any Modbus read or write activity. All the scenarios below are based on the diagram above.

Scenario 1: If the first MB_MASTER instruction reads 3 words of data from Modbus address 40001 on any valid Modbus slave, then the following occurs.

The word from address 40001 is stored in "Data".Array_1[1].

The word from address 40002 is stored in "Data".Array_1[2].

The word from address 40003 is stored in "Data".Array_1[3].

Scenario 2: If the first MB_MASTER instruction reads 4 words of data from Modbus address 40015 on any valid Modbus slave, then the following occurs.

The word from address 40015 is stored in "Data".Array_1[1].

The word from address 40016 is stored in "Data".Array_1[2].

The word from address 40017 is stored in "Data".Array_1[3].

The word from address 40018 is stored in "Data".Array_1[4].

Scenario 3: If the second MB_MASTER instruction reads 2 words of data from Modbus address 30033 on any valid Modbus slave, then the following occurs.

The word from address 30033 is stored in "Data".Array_2[1].

The word from address 30034 is stored in "Data".Array_2[2].

Scenario 4: If the third MB_MASTER instruction writes 4 words of data to Modbus address 40050 on any valid Modbus slave, then the following occurs.

The word from "Data".Array_3[1] is written to Modbus address 40050.

The word from "Data".Array_3[2] is written to Modbus address 40051.

The word from "Data".Array_3[3] is written to Modbus address 40052.

The word from "Data".Array_3[4] is written to Modbus address 40053.

Scenario 5: If the third MB_MASTER instruction writes 3 words of data to Modbus address 40001 on any valid Modbus slave, then the following occurs.

The word from "Data".Array_3[1] is written to Modbus address 40001.

The word from "Data".Array_3[2] is written to Modbus address 40002.

The word from "Data".Array_3[3] is written to Modbus address 40003.

Scenario 6: If the fourth MB_MASTER instruction uses a Mode 11 (retrieve valid message count) from any valid Modbus slave, the following occurs.

The count word is stored in "Data".Array_4[1].

Example bit reads and writes using word locations as DATA_PTR input

Table 6- 1 Scenario 7: Read 4 output bits starting at Modbus address 00001

MB_MASTER input values			Slave Modbus values	
MB_ADDR	27 (Slave example)		**00001**	**ON**
MODE	0 (Read)		**00002**	**ON**
DATA_ADDR	00001 (Outputs)		**00003**	**OFF**
DATA_LEN	4		**00004**	**ON**
DATA_PTR	"Data".Array_4		00005	ON
			00006	OFF
			00007	ON
			00008	OFF

"Data".Array_4[1] values after Modbus request	
MS (Most significant) Byte	LS (Least significant) Byte
xxxx-1011	xxxx-xxxx
x indicates that data is not changed	

Table 6- 2 Scenario 8: Read 12 output bits starting at Modbus address 00003

MB_MASTER input values			Slave Modbus values				
MB_ADDR	27 (Slave example)		00001	ON		**00010**	**ON**
MODE	0 (Read)		00002	ON		**00011**	**OFF**
DATA_ADDR	00003 (Outputs)		**00003**	**OFF**		**00012**	**OFF**
DATA_LEN	12		**00004**	**ON**		**00013**	**ON**
DATA_PTR	"Data".Array_4		**00005**	**ON**		**00014**	**OFF**
			00006	**OFF**		00015	ON
			00007	**ON**		00016	ON
			00008	**ON**		00017	OFF
			00009	**OFF**		00018	ON

"Data".Array_4[1] values after Modbus request	
MS Byte	LS Byte
1011-0110	xxxx-0100-
x indicates that data is not changed	

Table 6- 3 Scenario 9: Write 5 output bits starting at Modbus address 00001

MB_MASTER input values			Slave outputs before			Slave outputs after
MB_ADDR	27 (Slave example)		**00001**	**ON**		**OFF**
MODE	1 (Write)		**00002**	**ON**		ON
DATA_ADDR	00001 (Outputs)		**00003**	**OFF**		ON
DATA_LEN	5		**00004**	**ON**		**OFF**
DATA_PTR	"Data".Array_4		**00005**	**ON**		ON
			00006	OFF		Unchanged
			00007	ON		Unchanged
			00008	ON		Unchanged
			00009	OFF		Unchanged

"Data".Array_4[1] values for Modbus write request	
MS Byte	LS Byte
xxx1-0110	xxxxx-xxxx
x indicates that data is not used in the Modbus request	

Table 6- 4 Scenario 10: Read 22 output bits starting at Modbus address 00003

MB_MASTER input values			Slave Modbus values				
MB_ADDR	27 (Slave example)		00001	ON		00014	ON
MODE	0 (Read)		00002	ON		00015	OFF
DATA_ADDR	00003 (Outputs)		00003	OFF		00016	ON
DATA_LEN	22		00004	ON		00017	ON
DATA_PTR	"Data".Array_4		00005	ON		00018	OFF
			00006	OFF		00019	ON
			00007	ON		00020	ON
			00008	ON		00021	OFF
			00009	ON		00022	ON
			00010	OFF		00023	ON
			00011	OFF		00024	OFF
			00012	ON		00025	OFF
			00013	OFF		00026	ON

"Data".Array_4[1] values after Modbus request	
MS Byte	LS Byte
0111-0110	0110-1010

"Data".Array_4[2] values after Modbus request	
MS Byte	LS Byte
xx01-1011	xxxx-xxxx
x indicates that data is not changed	

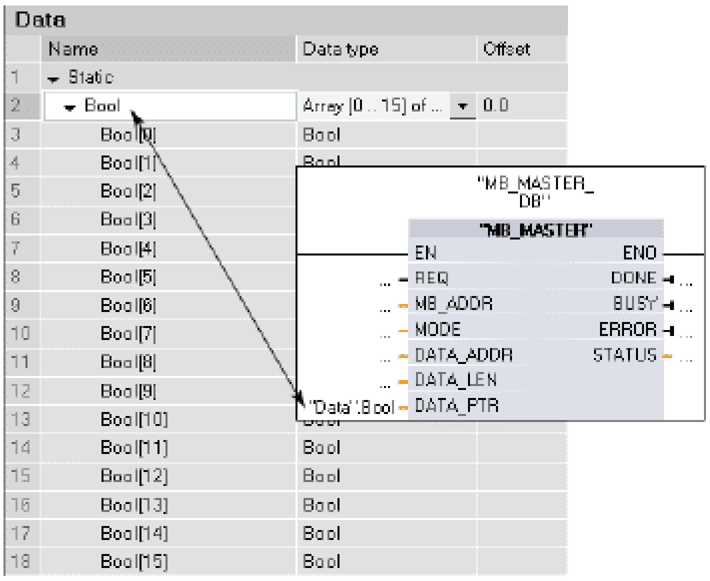
Example bit reads and writes using BOOL locations as DATA_PTR input

Although Modbus reads and writes to bit address locations can be handled through the use of word locations, DATA_PTR areas can also be configured as Boolean data types, structures or arrays to provide a direct one to one correlation for the first bit that is read or written by using a MB_MASTER instruction.

If you use Boolean arrays or structures, it is recommended that you make the data size a multiple of 8 bits (on byte boundaries). For example, when you create a Boolean array of 10 bits the STEP 7 Basic software will allocate 16 bits (2 bytes) in the global Data block for the 10 bits. Inside the data block, these would be stored as byte1 [xxxx xxxx] byte2 [---- --xx] where x indicates accessible data locations and – indicates locations that are inaccessible. Modbus requests of up to a length of 16 bits are allowed, but the upper 6 bits would be placed into byte 2 memory locations that are not referenced and are not accessible by your program.

Boolean areas can be created as an array of Boolean values or as a structure of Boolean variables. Both methods operate in an identical manner and differ only in how they are created and accessed in your program.

The global data block editor view below shows a single array of 16 Boolean values created with base 0. The array could also have been created as a base 1 array. The arrow shows how this array is associated with a MB_MASTER instruction.



Data

	Name	Data type	Offset
1	Static		
2	Bool	Array [0 .. 15] of ...	0.0
3	Bool[0]	Bool	
4	Bool[1]	Bool	
5	Bool[2]		
6	Bool[3]		
7	Bool[4]		
8	Bool[5]		
9	Bool[6]		
10	Bool[7]		
11	Bool[8]		
12	Bool[9]		
13	Bool[10]	Bool	
14	Bool[11]	Bool	
15	Bool[12]	Bool	
16	Bool[13]	Bool	
17	Bool[14]	Bool	
18	Bool[15]	Bool	

Scenarios 11 and 12 show the correspondence of Modbus addresses to Boolean array addresses.

Table 6- 5 Scenario 11: Write 5 output bits starting at Modbus address 00001

MB_MASTER input values			Slave outputs before		DATA_PTR data	Slave outputs after
MB_ADDR	27 (Slave example)		00001	ON	"Data".Bool[0]=FALSE	OFF
MODE	1 (Write)		00002	ON	"Data".Bool[1]=TRUE	ON
DATA_ADDR	00001 (Outputs)		00003	OFF	"Data".Bool[2]=TRUE	ON
DATA_LEN	5		00004	ON	"Data".Bool[3]-FALSE	OFF
DATA_PTR	"Data".Bool		00005	ON	"Data".Bool[4]=FALSE	OFF
			00006	OFF		Unchanged
			00007	ON		Unchanged
			00008	OFF		Unchanged

Table 6- 6 Scenario 12: Read 15 output bits starting at Modbus address 00004

MB_MASTER input values			Slave Modbus value		DATA_PTR data after
MB_ADDR	27 (Slave example)		00001	ON	
MODE	0 (Read)		00002	ON	
DATA_ADDR	00003 (Outputs)		**00003**	**OFF**	"Data".Bool[0]=FALSE
DATA_LEN	15		**00004**	**ON**	"Data".Bool[1]=TRUE
DATA_PTR	"Data".Bool		**00005**	**ON**	"Data".Bool[2]=TRUE
			00006	**OFF**	"Data".Bool[3]-FALSE
			00007	**ON**	Data".Bool[4]=TRUE
			00008	**ON**	Data".Bool[5]=TRUE
			00009	**ON**	Data".Bool[6]=TRUE
			00010	**OFF**	Data".Bool[7]=FALSE
			00011	**OFF**	Data".Bool[8]=FALSE
			00012	**ON**	Data".Bool[9]=TRUE
			00013	**OFF**	Data".Bool[10]=FALSE
			00014	**ON**	Data".Bool[11]=TRUE
			00015	**OFF**	Data".Bool[12]=FALSE
			00016	**ON**	Data".Bool[13]=TRUE
			00017	**ON**	Data".Bool[14]=TRUE
			00018	OFF	
			00019	ON	

Condition codes

STATUS value (W#16#....)	Description
0000	No error
80C8	The specified response timeout (refer to RCVTIME or MSGTIME) is 0.
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
8186	Invalid Modbus station address

STATUS value (W#16#....)	Description
8188	Invalid Mode value or write mode to read only slave address area
8189	Invalid Data Address value
818A	Invalid Data Length value
818B	• Invalid pointer to the local data source/destination: Size not correct
818C	Pointer to a type safe DB type DATA_PTR (must be a Classic DB type)
8200	Port is busy processing a transmit request

6.3.2.3 MB_SLAVE

The MB_SLAVE instruction allows your program to communicate as a Modbus slave using a port on the Point-to-Point (PtP) CM 1241 RS485 or CM 1241 RS232 module. A Modbus RTU master can issue a request and then your program responds via MB_SLAVE execution.

You must assign a unique Instance Data Block when you place the MB_SLAVE instruction in your program. This MB_SLAVE Instance Data Block name is used when you specify the MB_DB parameter on the MB_COMM_LOAD instruction.

Modbus communication function codes (1, 2, 4, 5, and 15) can read and write bits and words directly in the PLC Input Process Image and Output process Image. The following table shows the mapping of modbus addresses to the process image in the CPU.

MB_SLAVE 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 separate and unique Modbus holding register data block that you must create, before you can specify the MB_HOLD_REG parameter on the MB_SLAVE instruction. The following table shows the mapping of Modbus holding register to the MB_HOLD_REG DB address in the PLC.

MB_SLAVE Modbus functions				S7-1200	
Codes	Function	Data area	Address range	CPU DB data area	CPU DB address
03	Read words	Holding Register	40001 to 49999	MB_HOLD_REG	Words 1 to 9999
			400001 to 465535		Words 1 to 65535
06	Write word	Holding Register.	40001 to 49999	MB_HOLD_REG	Words 1 to 9999
			400001 to 465535		Words 1 to 65535
16	Write words	Holding Register	40001 to 49999	MB_HOLD_REG	Words 1 to 9999
			400001 to 465535		Words 1 to 65535

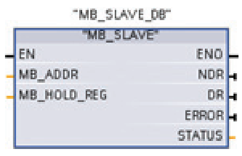
The following table shows the supported Modbus diagnostic functions.

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).

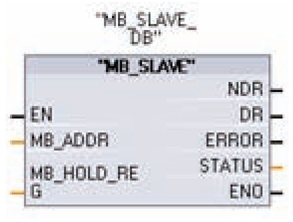
The MB_SLAVE supports broadcast write requests from any Modbus master as long as the request is for accessing valid locations.

Regardless of the validity of a request, the MB_SLAVE gives no response to a Modbus master as the result of a broadcast request.

LAD



FBD



Parameter	Parameter type	Data type	Description
MB_ADDR	IN	USINT	Modbus RTU Address (1 to 247): The station address of the Modbus slave.
MB_HOLD_REG	IN	VARIANT	Pointer to the Modbus Holding Register DB. The Holding Register DB must be a classic, global DB See the MB_HOLD_REG note below.
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

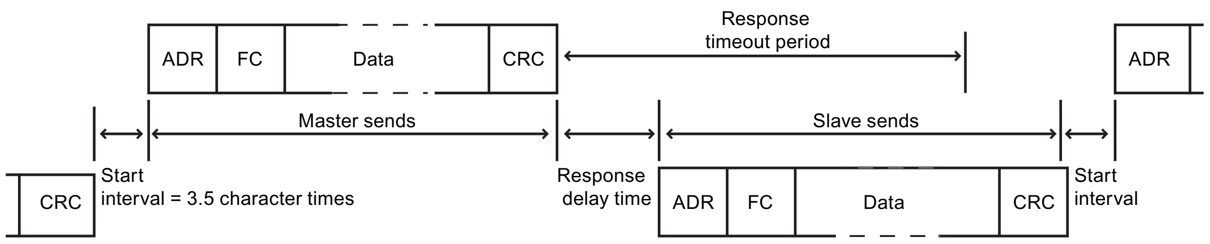
Parameter	Parameter type	Data type	Description
ERROR	OUT	BOOL	Error: • 0 – No error detected • 1 – Indicates that an error was detected and the error code supplied at parameter STATUS is valid.
STATUS	OUT	WORD	Error code

Modbus slave communication rules

- MB_COMM_LOAD must be executed to configure a port, before a MB_SLAVE instruction can communicate with that port.
- If a port is to respond as a slave to a Modbus master, then that port cannot be used by MB_MASTER. Only one instance of MB_SLAVE execution can be used with a given port.
- 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 must execute periodically at a rate that allows it to make a timely response to incoming requests from a Modbus master.
- You should call MB_SLAVE every scan from a program cycle OB.

Operation

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 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 (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.

MB_HOLD_REG parameter examples

MB_HOLD_REG is a pointer to the Modbus holding register Data block. This DB is used to hold data values that a Modbus master is allowed to access (read or write). You must create the data block and assign the data type structure that will be read from and written to, before it can be used with the MB_SLAVE instruction.

Note

The Modbus Holding Register data block must reference a global data block which was created with the Symbolic Access Only attribute box unchecked.

You must uncheck the "Symbolic address only" box when you add a new Data block to create a classic global DB type.

The holding registers can use these DB data structures:

- Standard array of words
- Named word structure
- Named complex structure

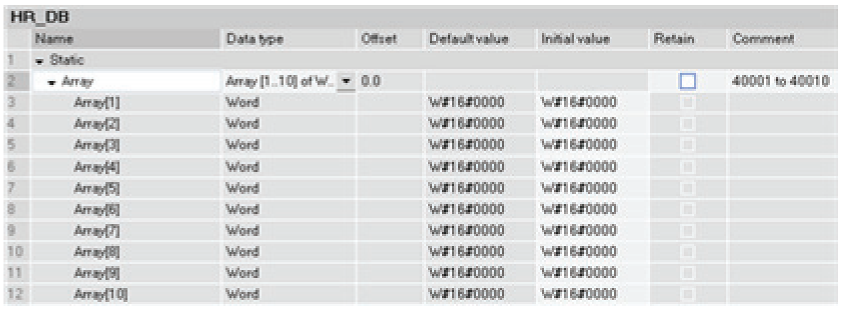
The following program examples show how to use the MB_HOLD_REG parameter to handle these DB data structures.

Example 1 - Standard array of words

This example holding register is an array of words. The data type assignments can be changed to other word size types (INT and UINT).

Advantages:	• This type of holding register structure is very fast and simple to create. • The program logic to access a data element is simplified.
•	
Disadvantages:	• Although you can programmatically reference each array element by the symbolic names ("HR_DB"."Array"[1] through "HR_DB"."Array"[10]), the names do not describe the internal function of the data. • The array can consist of only one data type. Type conversions may be required in a user program with rigid type control.

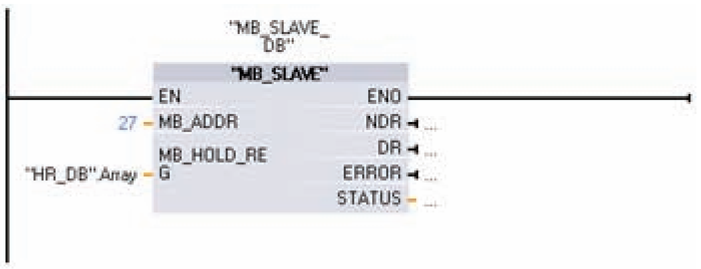
This is how an array of words structure would appear in the data block editor.



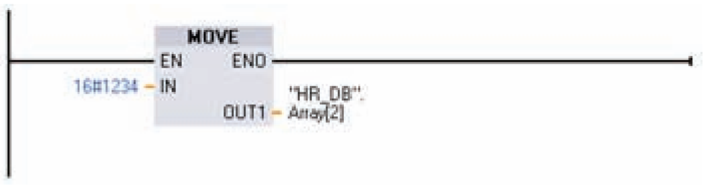
HR_DB

	Name	Data type	Offset	Default value	Initial value	Retain	Comment
1	Static						
2	Array	Array [1..10] of W..	0.0			☐	40001 to 40010
3	Array[1]	Word		W#16#0000	W#16#0000	☐	
4	Array[2]	Word		W#16#0000	W#16#0000	☐	
5	Array[3]	Word		W#16#0000	W#16#0000	☐	
6	Array[4]	Word		W#16#0000	W#16#0000	☐	
7	Array[5]	Word		W#16#0000	W#16#0000	☐	
8	Array[6]	Word		W#16#0000	W#16#0000	☐	
9	Array[7]	Word		W#16#0000	W#16#0000	☐	
10	Array[8]	Word		W#16#0000	W#16#0000	☐	
11	Array[9]	Word		W#16#0000	W#16#0000	☐	
12	Array[10]	Word		W#16#0000	W#16#0000	☐	

The image below shows how the array would be assigned to the MB_HOLD_REG input of an MB_SLAVE instruction.



Each element of the array can be accessed by symbolic name, as shown below. In this example, a new value is moved into the second element of the array which corresponds to Modbus address 40002.



Each of the words in the array, as defined in the data block, provides the MB_SLAVE instruction with Modbus holding register addresses. In this instance, since there are only 10 elements in the array, there are only 10 available Modbus holding register addresses usable by this MB_SLAVE instruction and accessible by a Modbus master.

The correlation of the array element names to Modbus addresses is shown below.

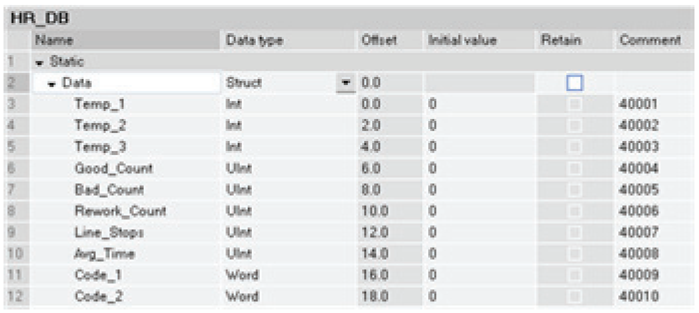
"HR_DB".Array[1]	Modbus address 40001
" HR_DB ". Array[2]	Modbus address 40002
" HR_DB ". Array[3]	Modbus address 40003
...	...
" HR_DB ". Array[9]	Modbus address 40009
" HR_DB ".Array [10]	Modbus address 40010

Example 2 - Named word structure

This example holding register is a series of words with descriptive symbolic names.

Advantages:	• Each structure element has a descriptive name with a specific data type assigned to it.
Disadvantages:	• It takes longer to create this type of structure than the standard array of words. • The elements require additional symbolic referencing when used in a user program. Where the first element of the simple array is referenced as "HR_DB".Array[0] the first element of this type is referenced "HR_DB".Data.Temp_1.

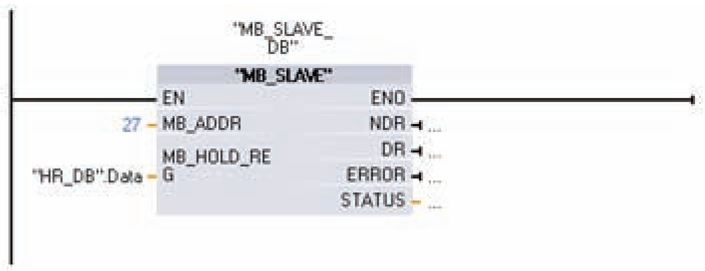
This is how a named word structure would appear in the data block editor. Each element has a unique name and can be a WORD, UINT, or INT.



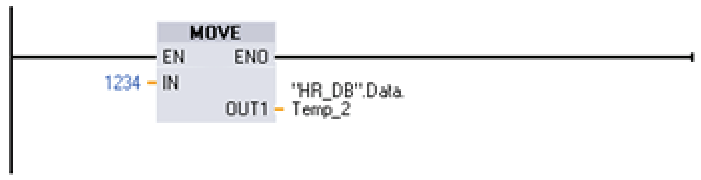
HR_DB

	Name	Data type	Offset	Initial value	Retain	Comment
1	▾ Static					
2	▾ Data	Struct	0.0		☐	
3	Temp_1	Int	0.0	0		40001
4	Temp_2	Int	2.0	0		40002
5	Temp_3	Int	4.0	0		40003
6	Good_Count	UInt	6.0	0		40004
7	Bad_Count	UInt	8.0	0		40005
8	Rework_Count	UInt	10.0	0		40006
9	Line_Stops	UInt	12.0	0		40007
10	Avg_Time	UInt	14.0	0		40008
11	Code_1	Word	16.0	0		40009
12	Code_2	Word	18.0	0		40010

The image below shows how the data structure above would be assigned to the MB_HOLD_REG input of an MB_SLAVE instruction in your program.



Each element of the array can be accessed by its symbolic name as shown below. In this example, a new value is moved into the second element of the array which corresponds to Modbus address 40002.



The correlation of the data element names to Modbus addresses is shown below.

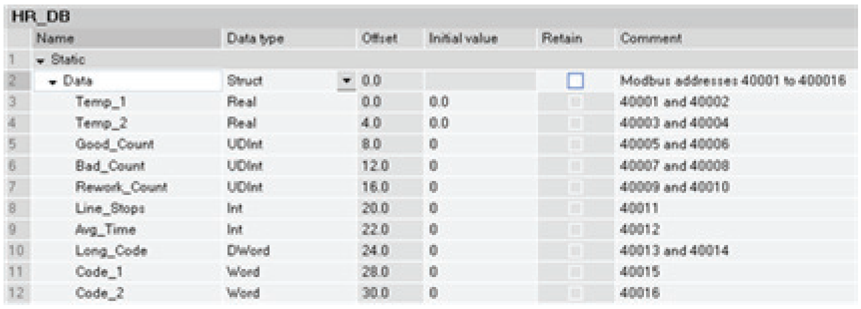
"HR_DB".Data.Temp_1	Modbus address 40001
"HR_DB".Data.Temp_2	Modbus address 40002
"HR_DB".Data.Temp_3	Modbus address 40003
"HR_DB".Data.Good_Count	Modbus address 40004
"HR_DB".Data.Bad_Count	Modbus address 40005
"HR_DB".Data.Rework_Count	Modbus address 40006
"HR_DB".Data.Line_Stops	Modbus address 40007
"HR_DB".Data.Avg_Time	Modbus address 40008
"HR_DB".Data.Code_1	Modbus address 40009
"HR_DB".Data.Code_2	Modbus address 40010

Example 3 - Named complex structure

This example holding register is a series of series of mixed data types with descriptive symbolic names.

Advantages:	• Each structure element has a descriptive name with a specific data type assigned to it. • It allows for the direct transfer of non-word based data types.
Disadvantages:	• It takes longer to create this type of structure than the standard array of words. • The Modbus master needs to be configured to accept the data that it will be receiving from the Modbus slave. As shown in the image below, Temp_1 is a 4 byte real value. The receiving master needs to be able to reassemble the 2 received words back into the real value that is expected. • The elements require additional symbolic referencing in your program. Where the first element of the simple array is referenced as "HR_DB".Array[0], the first element of this type is referenced as "HR_DB".Data.Temp_1.

This is how a named complex structure would appear in the data block editor. Each element has a unique name with multiple sizes and data types.



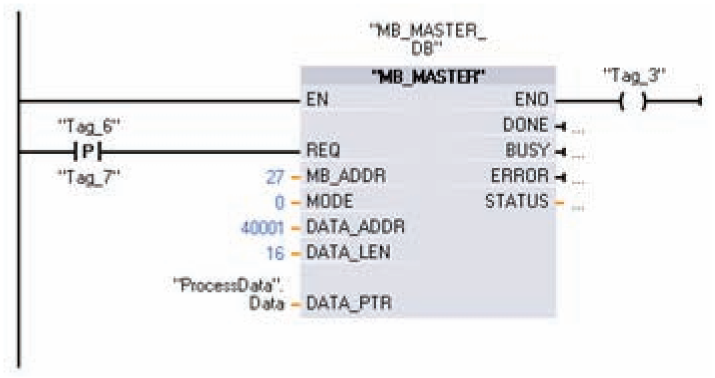
HR_DB

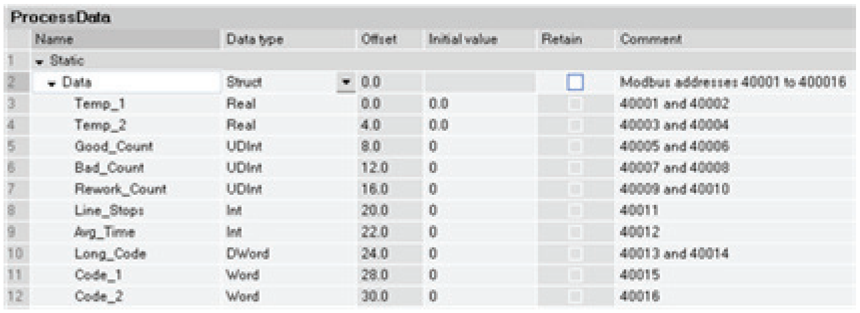
	Name	Data type	Offset	Initial value	Retain	Comment
1	▾ Static					
2	▾ Data	Struct	0.0		☐	Modbus addresses 40001 to 400016
3	Temp_1	Real	0.0	0.0	☐	40001 and 40002
4	Temp_2	Real	4.0	0.0	☐	40003 and 40004
5	Good_Count	UDInt	8.0	0	☐	40005 and 40006
6	Bad_Count	UDInt	12.0	0	☐	40007 and 40008
7	Rework_Count	UDInt	16.0	0	☐	40009 and 40010
8	Line_Stops	Int	20.0	0	☐	40011
9	Avg_Time	Int	22.0	0	☐	40012
10	Long_Code	DWord	24.0	0	☐	40013 and 40014
11	Code_1	Word	28.0	0	☐	40015
12	Code_2	Word	30.0	0	☐	40016

The correlation of the data element names to Modbus addresses is shown below.

"HR_DB".Data.Temp_1	Modbus addresses 40001 and 40002
"HR_DB".Data.Temp_2	Modbus addresses 40003 and 40004
"HR_DB".Data.Good_Count	Modbus addresses 40005 and 40006
"HR_DB".Data.Bad_Count	Modbus addresses 40007 and 40008
"HR_DB".Data.Rework_Count	Modbus addresses 40009 and 40010
"HR_DB".Data.Line_Stops	Modbus address 400011
"HR_DB".Data.Avg_Time	Modbus address 400012
"HR_DB".Data.Long_Code	Modbus address 40013 and 40014
"HR_DB".Data.Code_1	Modbus address 40015
"HR_DB".Data.Code_2	Modbus address 40016

Another S7-1200 CPU operating as a Modbus master can use the MB_Master instruction and an identical data structure to receive the block of data from the S7-1200 CPU operating as a Modbus slave. This Modbus Master instruction will copy all 16 words of data directly from the slave's HR_DB data block into the master's ProcessData data block, as shown below.





ProcessData

	Name	Data type	Offset	Initial value	Retain	Comment
1	▾ Static					
2	▾ Data	Struct	0.0		☐	Modbus addresses 40001 to 400016
3	Temp_1	Real	0.0	0.0	☐	40001 and 40002
4	Temp_2	Real	4.0	0.0	☐	40003 and 40004
5	Good_Count	UDInt	8.0	0	☐	40005 and 40006
6	Bad_Count	UDInt	12.0	0	☐	40007 and 40008
7	Rework_Count	UDInt	16.0	0	☐	40009 and 40010
8	Line_Stops	Int	20.0	0	☐	40011
9	Avg_Time	Int	22.0	0	☐	40012
10	Long_Code	DWord	24.0	0	☐	40013 and 40014
11	Code_1	Word	28.0	0	☐	40015
12	Code_2	Word	30.0	0	☐	40016

A series of Modbus master Data_PTR data block locations can be used to transfer the same or different structures from multiple Modbus slaves.

Condition codes

STATUS value (W#16#....)	Description
80C8	The specified response timeout (refer to RCVTIME or MSGTIME) is 0
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
8186	Invalid Modbus station address
8187	Invalid pointer to MB_HOLD_REG DB
818C	Pointer to a type safe DB type MB_HOLD_REG DB (must be a Classic DB type)

STATUS value (W#16#....)	Description	
	Response code sent to Modbus master (B#16#..)	
8380	No response	CRC error
8381	01	Function code not supported
8382	No response	Data length error
8383	02	Data address error
8384	03	Data value error
8385	03	Data diagnostic code value not supported (function code 08)

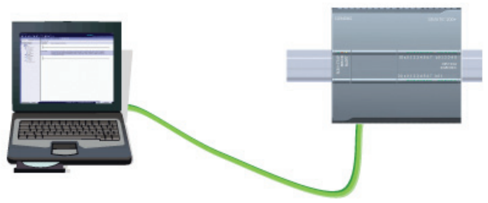
PROFINET 7

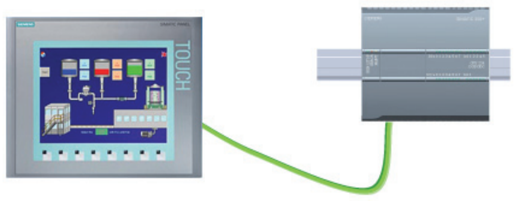
The S7-1200 CPU has an integrated PROFINET port which supports both Ethernet and TCP/IP-based communications standards. The following application protocols are supported by the S7-1200 CPU:

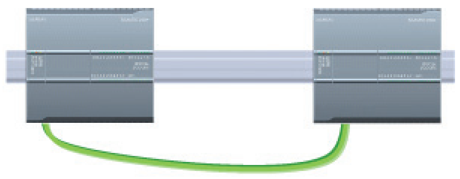
- Transport Control Protocol (TCP)
- ISO on TCP (RFC 1006)

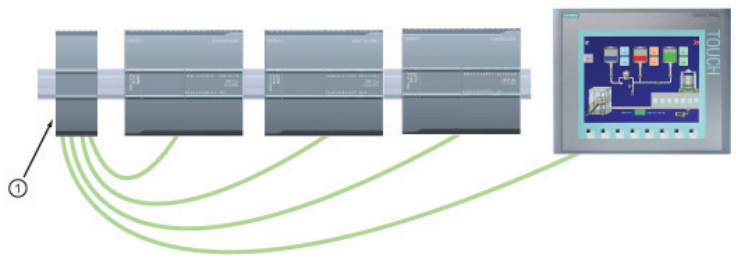
The S7-1200 CPU can communicate with other S7-1200 CPUs, with the STEP 7 Basic programming device, with HMI devices, and with non-Siemens devices using standard TCP communications protocols. There are two ways to communicate using PROFINET:

- Direct connection: Use direct communication when you are using a programming device, HMI, or another CPU connected to a single CPU.
- Network connection: Use network communications when you are connecting more than two devices (for example, CPUs, HMIs, programming devices, and non-Siemens devices).


Direct connection: Programming device connected to S7-1200 CPU


Direct connection: HMI connected to S7-1200 CPU


Direct connection: An S7-1200 CPU connected to another S7-1200 CPU



Network connection: More than two devices connected together, using a CSM1277 Ethernet switch ①

An Ethernet switch is not required for a direct connection between a programming device or HMI and a CPU. An Ethernet switch is required for a network with more than two CPUs or HMI devices. The rack-mounted Siemens CSM1277 4-port Ethernet switch can be used to connect your CPUs and HMI devices. The PROFINET port on the S7-1200 CPU does not contain an Ethernet switching device.

Maximum number of connections for the PROFINET port

The PROFINET port on the CPU supports the following simultaneous communication connections.

- 3 connections for HMI to CPU communication
- 1 connection for programming device (PG) to CPU communication
- 8 connections for S7-1200 program communication using the T-block instructions (TSEND_C, TRCV_C, TCON, TDISCON, TSEN, TRCV)
- 3 connections for a passive S7-1200 CPU communicating with an active S7 CPU
  - The active S7 CPU uses GET and PUT instructions (S7-300 and S7-400) or ETHx_XFER instructions (S7-200).
  - An active S7-1200 communication connection is only possible with the T-block instructions.

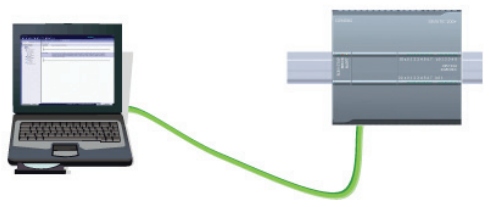
Restricted TSAPs or port numbers for passive ISO and TCP communication

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
- TCP port (passive): 5001, 102, 123, 20, 21, 25, 34962, 34963, 34964, 80

7.2 Communication with a programming device

A CPU can communicate with a STEP 7 Basic programming device on a network.



Consider the following when setting up communications between a CPU and a programming device:

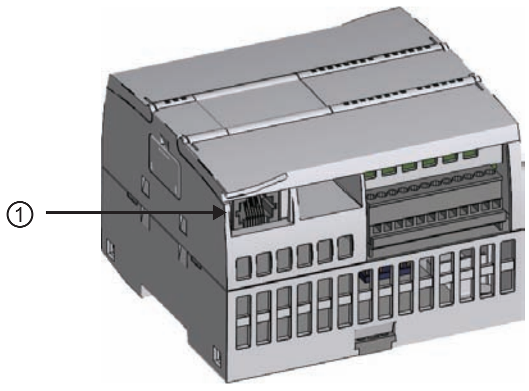
- Configuration/Setup: Hardware configuration is required.
- No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

7.2.1 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 26).
2. Plug the Ethernet cable into the PROFINET port shown below.
3. Connect the Ethernet cable to the programming device.



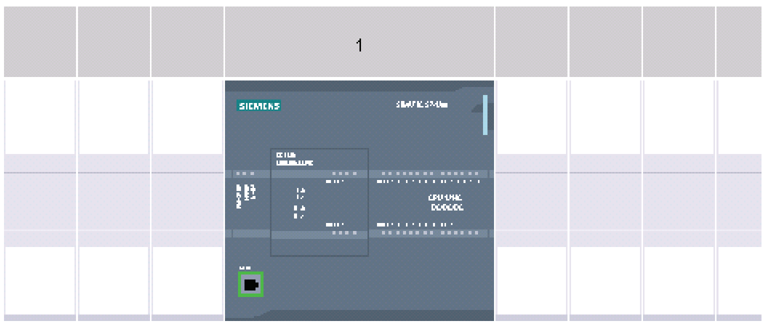
① PROFINET port

An optional strain relief is available to strengthen the PROFINET connection.

7.2.2 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 70) into the rack. In the project below, a CPU is shown in the "Device View" of the TIA Portal.



7.2.3 Assigning Internet Protocol (IP) addresses

7.2.3.1 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.

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 two 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 two 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

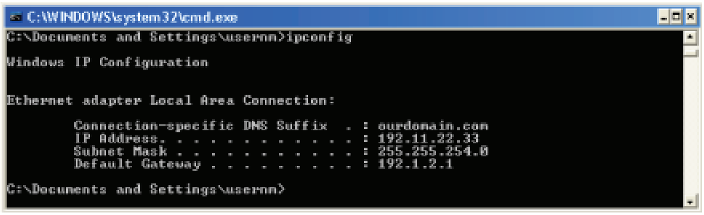
Dynamic Host Configuration Protocol (DHCP) automatically assigns an IP address to your programming device upon power up from the DHCP server.

Checking the IP address of your programming device using the "ipconfig" and "ipconfig /all" commands

You can also check the IP address of your programming device, and, if applicable, the IP address of your IP router (Gateway) with the following menu selections:

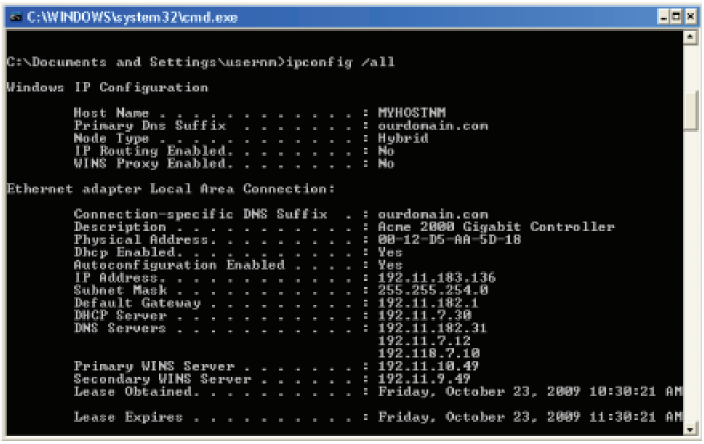
- "Start" button (on your desktop)
- "Run"

In the "Run" dialog, in the "Open" field, enter "cmd" and click the "OK" button. In the "C:\WINDOWS\system32\cmd.exe" dialog that is displayed, enter the command "ipconfig". An example result is shown below:



```
C:\Documents and Settings\usernm>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : ourdomain.com
        IP Address. . . . . . . . . . . . : 192.11.22.33
        Subnet Mask . . . . . . . . . . . : 255.255.254.0
        Default Gateway . . . . . . . . . : 192.1.2.1

C:\Documents and Settings\usernm>
```

Further information can be displayed with an "ipconfig /all" command. Your programming device's adapter card type and Ethernet (MAC) address can be found here:



```

C:\Documents and Settings\usernm>ipconfig /all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : MYHOSTNM
        Primary Dns Suffix  . . . . . . . : ourdomain.com
        Node Type . . . . . . . . . . . . : Hybrid
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : ourdomain.com
        Description . . . . . . . . . . . : Acme 2000 Gigabit Controller
        Physical Address. . . . . . . . . : 00-12-D5-AA-5D-18
        Dhcp Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : 192.11.183.136
        Subnet Mask . . . . . . . . . . . : 255.255.254.0
        Default Gateway . . . . . . . . . : 192.11.182.1
        DHCP Server . . . . . . . . . . . : 192.11.7.30
        DNS Servers . . . . . . . . . . . : 192.11.182.31
                                            192.11.7.12
                                            192.118.7.10
        Primary WINS Server . . . . . . . : 192.11.10.49
        Secondary WINS Server . . . . . . : 192.11.9.49
        Lease Obtained. . . . . . . . . . : Friday, October 23, 2009 10:30:21 AM

        Lease Expires . . . . . . . . . . : Friday, October 23, 2009 11:30:21 AM
```

Assigning an IP address to a CPU

You can use one of the following two methods to assign IP addresses to a CPU:

- Assign an IP address online
- Configure an IP address in your project

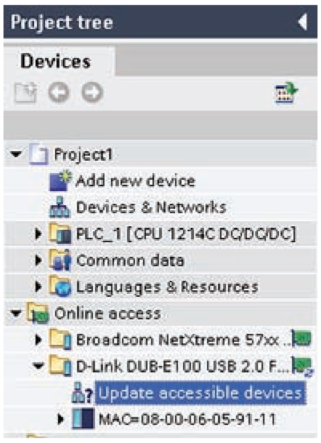
7.2.3.2 Assigning an IP address online

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

Use the following procedure to assign an IP address online:

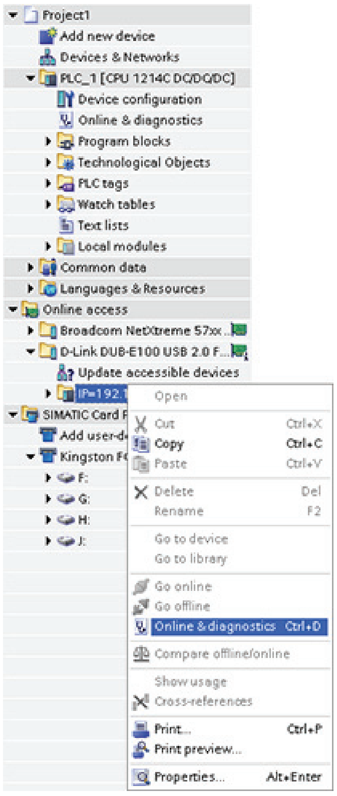
1. In the "Project tree," verify that no IP address is assigned to the CPU, with the following menu selections:

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



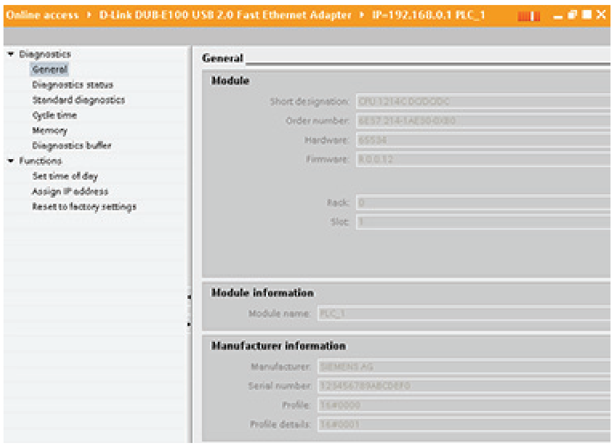
2. In the "Project tree," make the following menu selections:

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



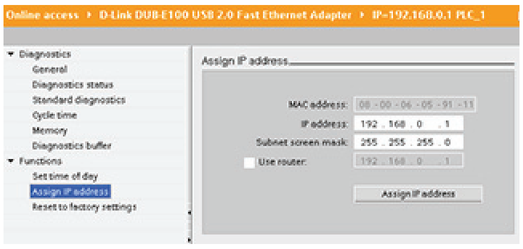
3. In the "Online & diagnostics" dialog, make the following menu selections:

- "Functions"
- "Assign IP address"



Short designation: CPU 1214C DC/DC/DC
Order number: 6ES7 214-1AE30-0XB0
Hardware: 65534
Firmware: R0.0.12
Rack: 0
Slot: 1
Module name: PLC_1
Manufacturer: SIEMENS AG
Serial number: 123456789ABCDEF0
Profile: 16#0000
Profile details: 16#0001

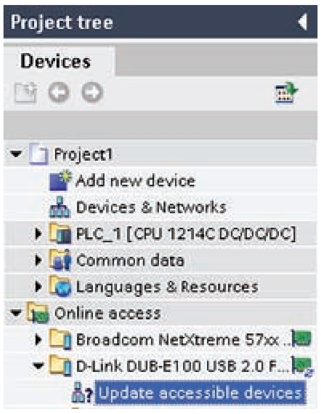
4. In the "IP address" field, enter your new IP address.



MAC address: 08 - 00 - 06 - 05 - 91 - 11
IP address: 192 . 168 . 0 . 1
Subnet screen mask: 255 . 255 . 255 . 0
Use router: 192 . 168 . 0 . 1

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

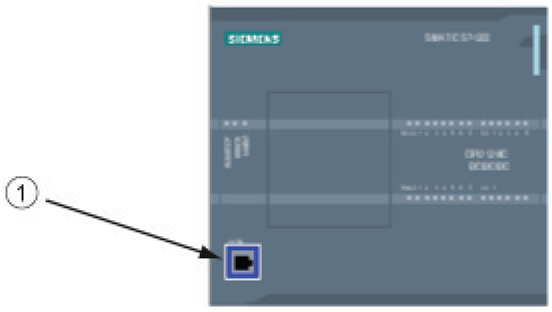
- "Online access"
- <Adapter for the network in which the device is located>
- "Update accessible devices"


Project tree
Devices
Project1
Add new device
Devices & Networks
PLC_1 [CPU 1214C DC/DC/DC]
Common data
Languages & Resources
Online access
Broadcom NetXtreme 57xx ...
D-Link DUB-E100 USB 2.0 F...
Update accessible devices

7.2.3.3 Configuring an IP address in your project

Configuring the PROFINET interface

After you configure the rack with the CPU (Page 211) , you can configure parameters for the PROFINET interface. To do so, click the green PROFINET box on the CPU to select the PROFINET port. 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.

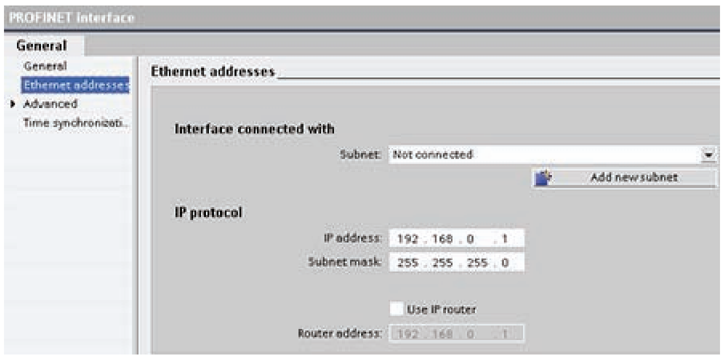
Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet tend to be located in close physical proximity to each other on a Local Area Network (LAN). A mask (known as the subnet mask or network mask) defines the boundaries of an IP subnet.

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.



PROFINET interface

General
- General
- Ethernet addresses
- Advanced
- Time synchronizati...

Ethernet addresses

Interface connected with

Subnet: Not connected

Add new subnet

IP protocol

IP address: 192 . 168 . 0 . 1
Subnet mask: 255 . 255 . 255 . 0

Use IP router

Router address: 192 . 168 . 0 . 1

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

Note

The CPU does not have a pre-configured IP address. You must manually assign an IP address for the CPU. If your CPU is connected to a router on a network, you must also enter the router's IP address. All IP addresses are configured when you download the project.

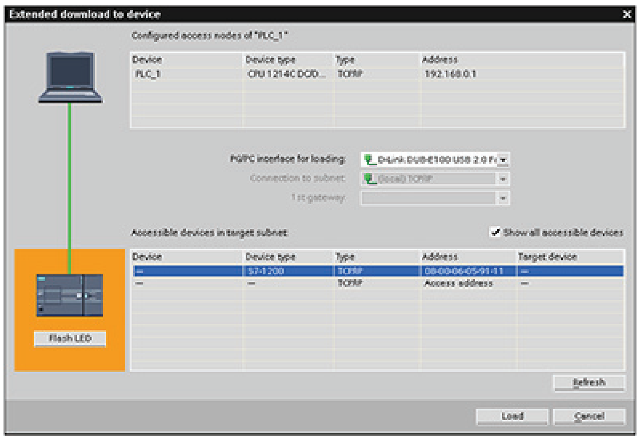
Refer to "Assigning IP addresses to programming and network devices" for more information.

The following table defines the 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

7.2.4 Testing the PROFINET network

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


Extended download to device

Configured access nodes of "PLC_1"

Device	Device type	Type	Address
PLC_1	CPU 1214C DC/D...	TCP/IP	192.168.0.1

PG/PC interface for loading: D-Link DUB-E100 USB 2.0 F...
Connection to subnet: (local) TCP/IP
1st gateway:

Accessible devices in target subnet:
✓ Show all accessible devices

Device	Device type	Type	Address	Target device
—	S7-1200	TCP/IP	08-00-06-05-91-11	—
—	—	TCP/IP	Access address	—

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 "Assigning an IP address online" for this step-by-step procedure.

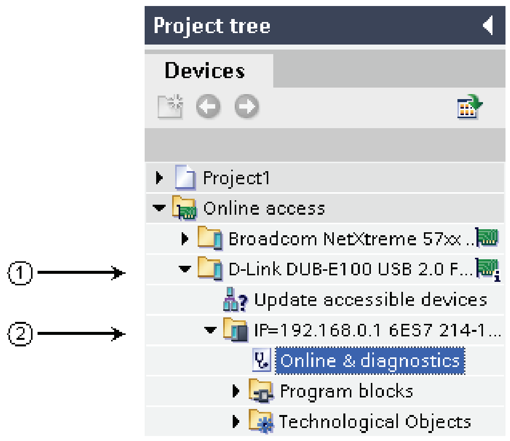
To assign an IP address in your project, you must configure the IP address in the Device configuration, save the configuration, and download it to the PLC. Refer to "Configuring an IP address for your project" for more information.

Note

If you have assigned IP addresses online you can change online-assigned IP addresses using the online or offline hardware configuration method.

If you have assigned IP addresses in offline hardware configuration, you can only change project-assigned IP addresses using the offline hardware configuration method.

Use "Online access" to display the connected CPU's IP address as shown below.



Project tree

Devices

- Project1
- Online access
  - Broadcom NetXtreme 57xx ...
  - D-Link DUB-E100 USB 2.0 F...
    - Update accessible devices
    - IP=192.168.0.1 6ES7 214-1...
      - Online & diagnostics
      - Program blocks
      - Technological Objects

①	Second of two Ethernet networks on this programming device
②	IP address of the only S7-1200 CPU on this Ethernet network

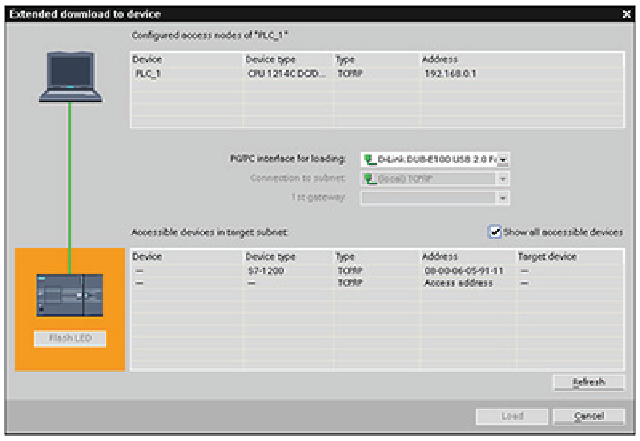
---

Note

All configured networks of the programming device are displayed. You must select the correct network to display the required S7-1200 CPU's IP address.

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 and IP addresses, check the "Show all accessible devices" checkbox.



Extended download to device

Configured access nodes of "PLC_1"

Device	Device type	Type	Address
PLC_1	CPU 1214C DC/D...	TCP/IP	192.168.0.1

PG/PC interface for loading: D-Link DUB-E100 USB 2.0 F...
Connection to subnet: (local) TCP/IP
1st gateway:

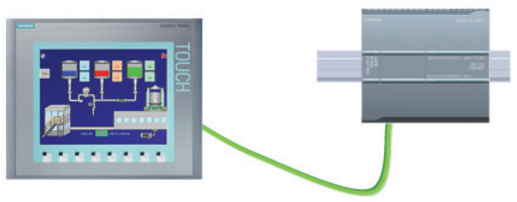
Accessible devices in target subnet:

✓ Show all accessible devices

Device	Device type	Type	Address	Target device
—	S7-1200	TCP/IP	08-00-06-05-91-11	—
—	—	TCP/IP	Access address	—

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.

7.3 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 Siemens 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

Note

WinCC Basic and STEP 7 Basic are components of the TIA Portal. Refer to WinCC Basic for more information on configuring the HMI.

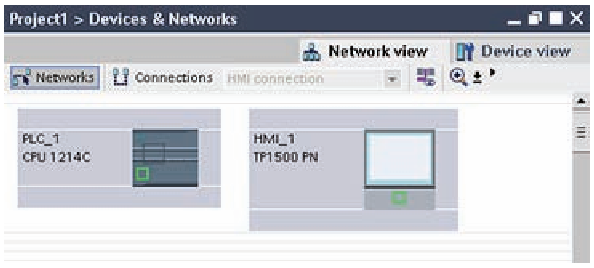
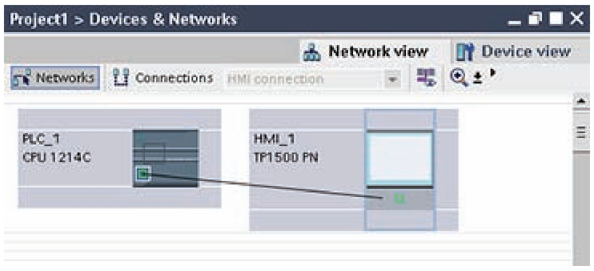
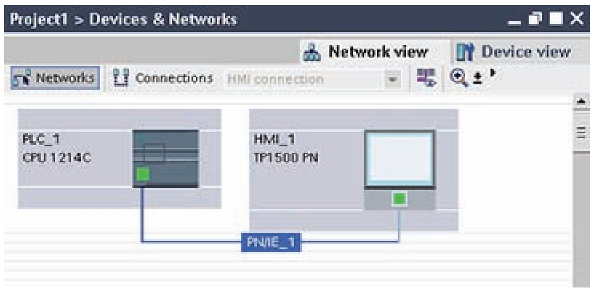
Required steps in configuring communications between an HMI and a CPU

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 211) for more information.
2	Configuring the devices Refer to "Communication with a programming device: Configuring the devices" (Page 211) 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 an HMI and a CPU" (Page 222) 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 "Communication with a programming device: Configuring an IP address in your project" (Page 217) for more information.
5	Testing the PROFINET network You must download the configuration for each CPU. Refer to "Communication with a programming device: Testing the PROFINET network (Page 219) for more information.

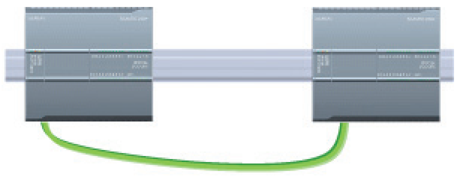
7.3.1 Configuring the logical network connections between an HMI and a CPU

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 the Ethernet connection, select the green (Ethernet) box on the CPU. Drag a line to the Ethernet box on the HMI device. Release the mouse button and your Ethernet connection is joined.

Action	Result
Select "Network view" to display the devices to be connected.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C HMI_1 TP1500 PN
Select the port on one device and drag the connection to the port on the second device.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C HMI_1 TP1500 PN
Release the mouse button to create the network connection.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C HMI_1 TP1500 PN PN/IE_1

7.4 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.

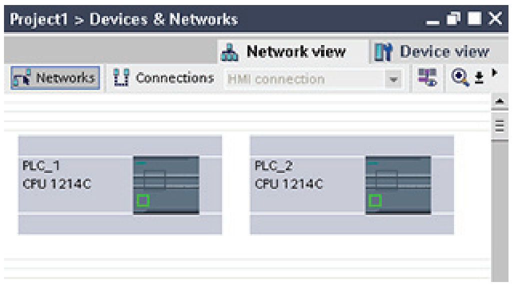
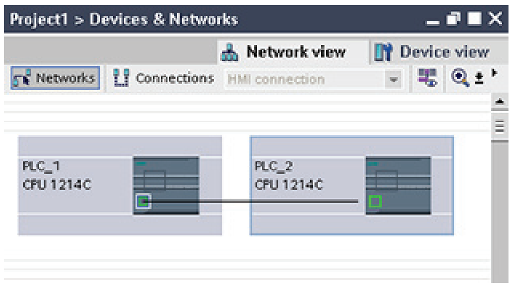
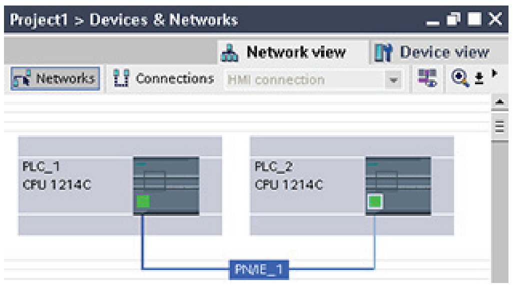
Required steps in configuring communications between two CPUs

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" for more information.
2	Configuring the devices You must configure two projects with a CPU in each project. Refer to "Communication with a programming device: Configuring the devices" for more information.
3	Configuring the logical network connections between two CPUs Refer to "Configuring communications between two CPUs: Configuring the logical network connections between two CPUs" (Page 224) for more information.
4	Configuring an IP address in yourproject Use the same configuration process; however, you must configure IP addresses for two CPUs (for example, PLC_1 and PLC_2). Refer to "Communication with a programming device: Configuring an IP address in your project" for more information.
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 225) for more information.
6	Testing the PROFINET network You must download the configuration for each CPU. Refer to "Configuring communications between a programming device and a CPU: Testing the PROFINET network" for more information.

7.4.1 Configuring the logical network connections between two CPUs

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 the PROFINET connection, select the green (PROFINET) box on the first PLC. Drag a line to the PROFINET box on the second PLC. Release the mouse button and your PROFINET connection is joined.

Action	Result
Select "Network view" to display the devices to be connected.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C PLC_2 CPU 1214C
Select the port on one device and drag the connection to the port on the second device.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C PLC_2 CPU 1214C
Release the mouse button to create the network connection.	 Project1 > Devices & Networks Network view Device view Networks Connections HMI connection PLC_1 CPU 1214C PLC_2 CPU 1214C PN/IE_1

7.4.2 Configuring transmit (send) and receive parameters

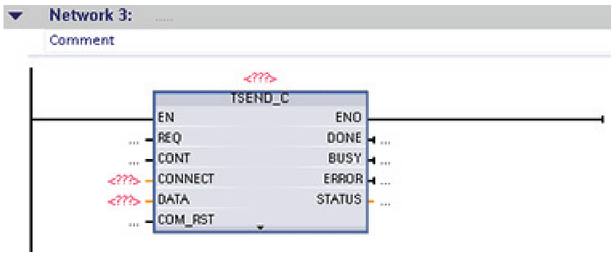
Transmission block (T-block) communications are used to establish connections between two CPUs. Before the CPUs can engage in PROFINET communications, you must configure parameters for transmitting (or sending) messages and receiving messages. These parameters dictate how communications operate when messages are being transmitted to or received from a target device.

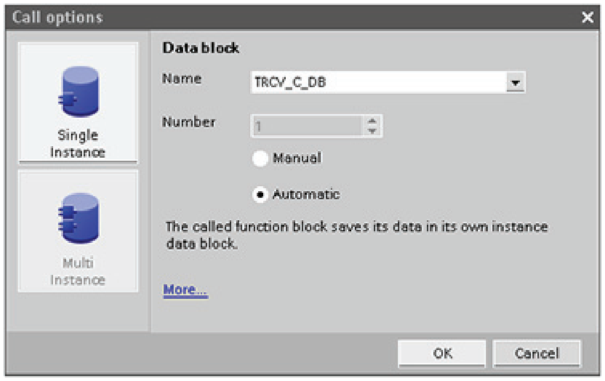
7.4.2.1 Configuring the TSEND_C instruction transmit (send) parameters

TSEND_C instruction

The TSEND_C instruction (Page 154) 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 Basic, 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 "Extended Instructions". The instruction is displayed, along with the Call options dialog where you assign a DB for storing the parameters of the TSEND_C instruction.




Call options

Single Instance

Multi Instance

Data block

Name: TRCV_C_DB

Number: 1

Manual

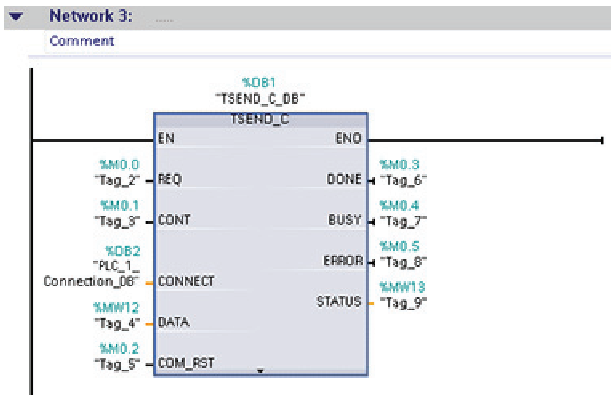
● Automatic

The called function block saves its data in its own instance data block.

More...

OK | Cancel

You can assign tag memory locations to the inputs and outputs, as shown in the following figure.



Configuring General parameters

You specify the communication 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 Connection parameters

Every CPU has an integrated PROFINET port, which supports standard PROFINET communications. The supported Ethernet protocols are described in the following two connection types:

Protocol	Protocol Name	Usage
RFC 1006	ISO on TCP	Message fragmentation and re-assembly
TCP	Transport Controool Protocol	Transport of frames

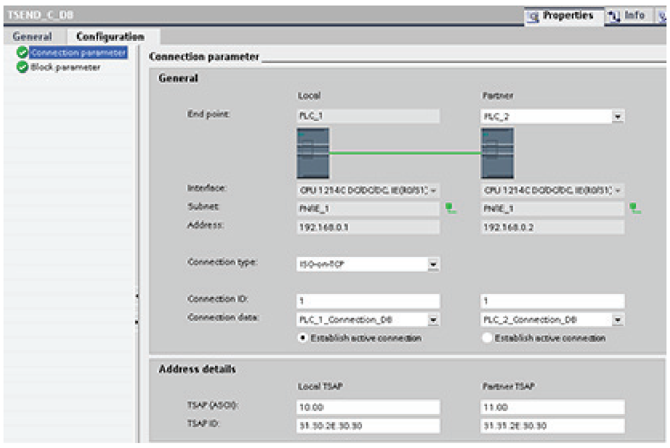
ISO on TCP (RFC 1006)

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.

UsingTransport 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.

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.



Connection parameter

General

	Local	Partner
End point:	PLC_1	PLC_2
Interface:	CPU 1214C DC/DC/DC, IE(R0/S1)	CPU 1214C DC/DC/DC, IE(R0/S1)
Subnet:	PN/IE_1	PN/IE_1
Address:	192.168.0.1	192.168.0.2
Connection type:	ISO-on-TCP	
Connection ID:	1	1
Connection data:	PLC_1_Connection_DB	PLC_2_Connection_DB
	● Establish active connection	○ Establish active connection

Address details

	Local TSAP	Partner TSAP
TSAP (ASCII):	10.00	11.00
TSAP ID:	31.30.2E.30.30	31.31.2E.30.30

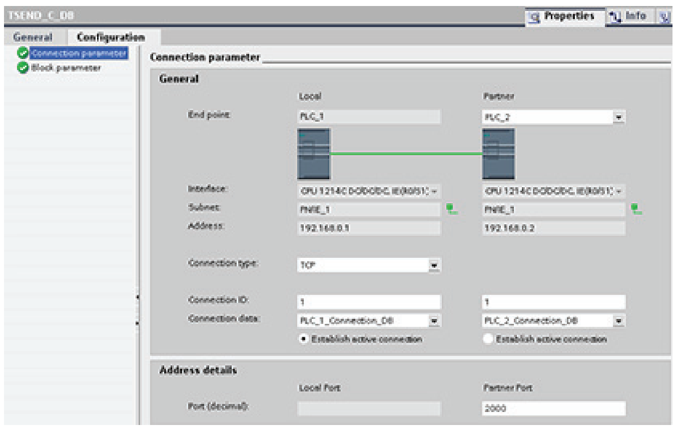
Parameter	Definition
General	
End point: Partner	Name assigned to the partner (receiving) CPU
Interface	Name assigned to the interfaces
Subnet	Name assigned to the subnets
Address	Assigned IP Addresses
Connection type	Type of Ethernet protocol
Connection ID	ID number
Connection data	Local and Partner CPU data storage location
Active connection setup	Radio button to select Local or Partner CPU as the active connection
Address details	
TSAP[1] (ASCII)	Local and Partner CPU TSAPs in ASCII format
TSAP ID	Local and Partner CPU TSAPs in hexadecimal format

[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 Control Protocol (TCP)

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
  - 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.


TSEND_C_DB

Connection parameter

General

	Local	Partner
End point:	PLC_1	PLC_2
Interface:	CPU 1214C DC/DC/DC, IE(R0/S1)	CPU 1214C DC/DC/DC, IE(R0/S1)
Subnet:	PN/IE_1	PN/IE_1
Address:	192.168.0.1	192.168.0.2
Connection type:	TCP	
Connection ID:	1	1
Connection data:	PLC_1_Connection_DB	PLC_2_Connection_DB
	● Establish active connection	○ Establish active connection

Address details

	Local Port	Partner Port
Port (decimal):		2000

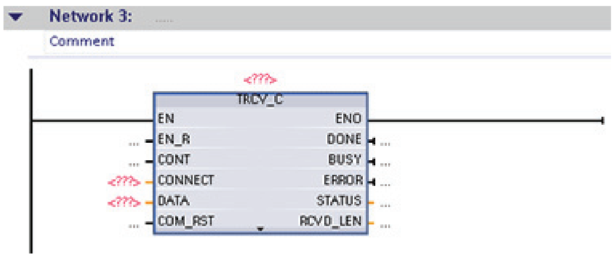
Parameter	Definition
General	
End point: Partner	Name assigned to the partner (receiving) CPU
Interface	Name assigned to the interfaces
Subnet	Name assigned to the subnets
Address	Assigned IP Addresses
Connection type	Type of Ethernet protocol
Connection ID	ID number
Connection data	Local and Partner CPU data storage location
Active connection setup	Radio button to select Local or Partner CPU as the active connection
Address details	
Port (decimal)	Partner CPU Port in decimal format

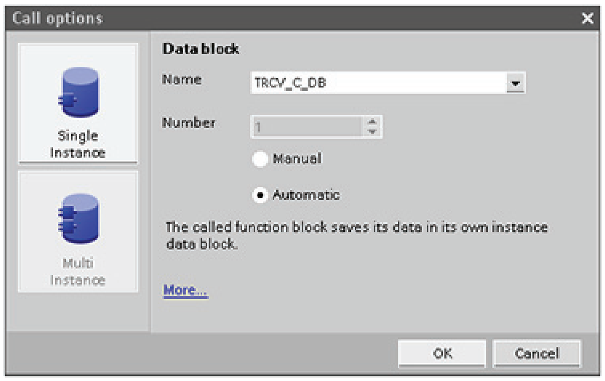
7.4.2.2 Configuring the TRCV_C instruction receive parameters

TRCV_C instruction

The TRCV_C instruction (Page 154) 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 Basic, you can configure how a TRCV_C instruction receives data. To begin, insert the instruction into the program from "Communications" folder in the "Extended Instructions". The instruction is displayed, along with the Call options dialog where you assign a DB for storing the parameters of the TRCV_C instruction.




Call options

Single Instance

Multi Instance

Data block

Name: TRCV_C_DB

Number: 1

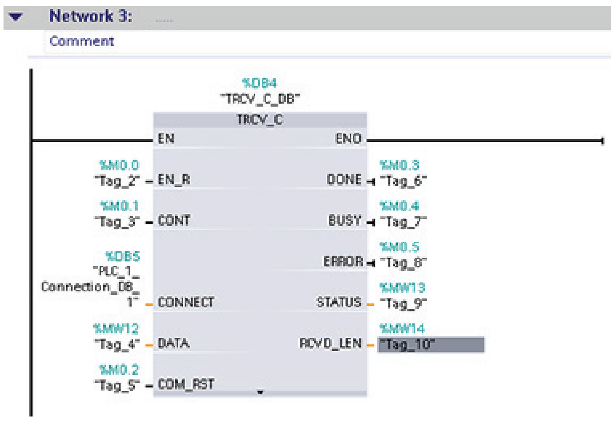
Manual [✓ Automatic]

The called function block saves its data in its own instance data block.

More...

OK Cancel

You can assign tag memory locations to the inputs and outputs, as shown as in the following figure.



Configuring the General parameters

You specify the communication 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.

Configuring the Connection parameters

Every CPU has an integrated PROFINET port, which supports standard PROFINET communications. The supported Ethernet protocols are described in the following two connection types:

Protocol	Protocol Name	Usage
RFC 1006	ISO on TCP	Message fragmentation and re-assembly
TCP	Transport Control Protocol	Transport of frames

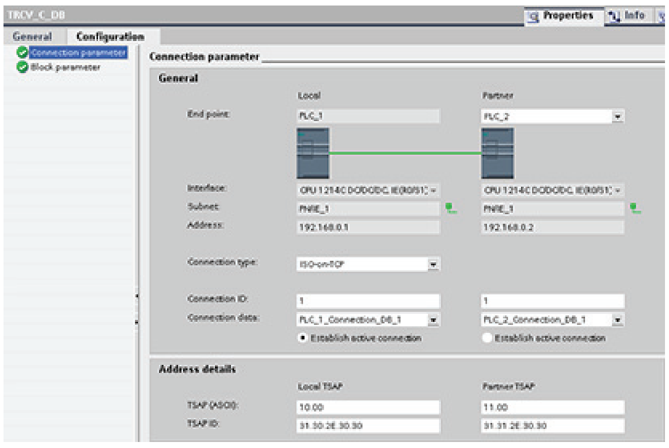
ISO on TCP (RFC 1006)

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.

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.



Connection parameter

General

	Local	Partner
End point:	PLC_1	PLC_2
Interface:	CPU 1214C DC/DC/DC, IE(R0/S1)	CPU 1214C DC/DC/DC, IE(R0/S1)
Subnet:	PN/IE_1	PN/IE_1
Address:	192.168.0.1	192.168.0.2
Connection type:	ISO-on-TCP	
Connection ID:	1	1
Connection data:	PLC_1_Connection_DB_1	PLC_2_Connection_DB_1
	[● Establish active connection]	○ Establish active connection

Address details

	Local TSAP	Partner TSAP
TSAP (ASCII):	10.00	11.00
TSAP ID:	31.30.2E.30.30	31.31.2E.30.30

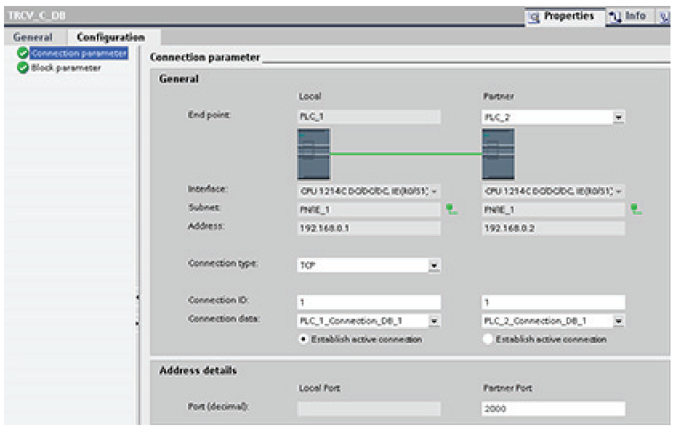
Parameter	Definition
General	
End point: Partner	Name assigned to the partner (receiving) CPU
Interface	Name assigned to the interfaces
Subnet	Name assigned to the subnets
Address	Assigned IP Addresses
Connection type	Type of Ethernet protocol
Connection ID	ID number
Connection data	Local and Partner CPU data storage location
Active connection setup	Radio button to select Local or Partner CPU as the active connection
Address details	
TSAP[1] (ASCII)	Local and Partner CPU TSAPs in ASCII format
TSAP ID	Local and Partner CPU TSAPs in hexadecimal format

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 Control Protocol (TCP)

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
  - 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.



Connection parameter

General

	Local	Partner
End point:	PLC_1	PLC_2
Interface:	CPU 1214C DC/DC/DC, IE(R0/S1)	CPU 1214C DC/DC/DC, IE(R0/S1)
Subnet:	PN/IE_1	PN/IE_1
Address:	192.168.0.1	192.168.0.2
Connection type:	TCP	
Connection ID:	1	1
Connection data:	PLC_1_Connection_DB_1	PLC_2_Connection_DB_1
	● Establish active connection	○ Establish active connection

Address details

	Local Port	Partner Port
Port (decimal):		2000

Parameter	Definition
General	
End point: Partner	Name assigned to the partner (receiving) CPU
Interface	Name assigned to the interfaces
Subnet	Name assigned to the subnets
Address	Assigned IP Addresses
Connection type	Type of Ethernet protocol
Connection ID	ID number
Connection data	Local and Partner CPU data storage location
Active connection setup	Radio button to select Local or Partner CPU as the active connection
Address details	
Port (decimal)	Local CPU Port in decimal format

7.5 Reference Information

7.5.1 Locating the Ethernet (MAC) address on the CPU

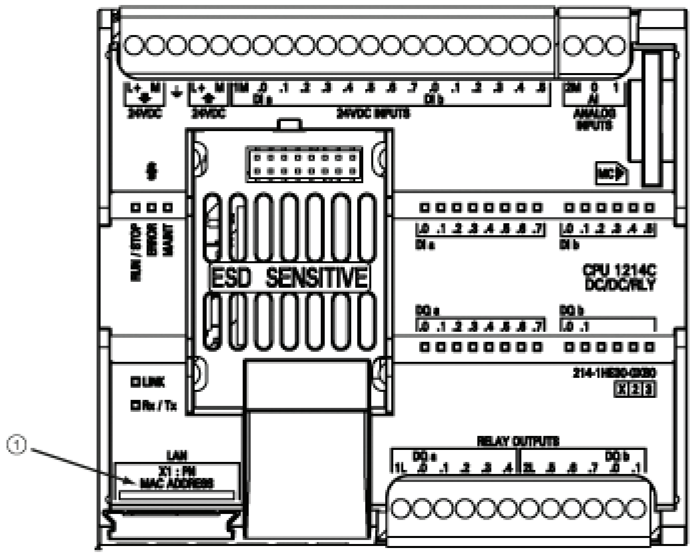
In PROFINET networking, a Media Access Control address (MAC address) is an identifier assigned to adapter cards by the manufacturer for identification. A MAC address usually encodes the manufacturer's registered identification number.

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

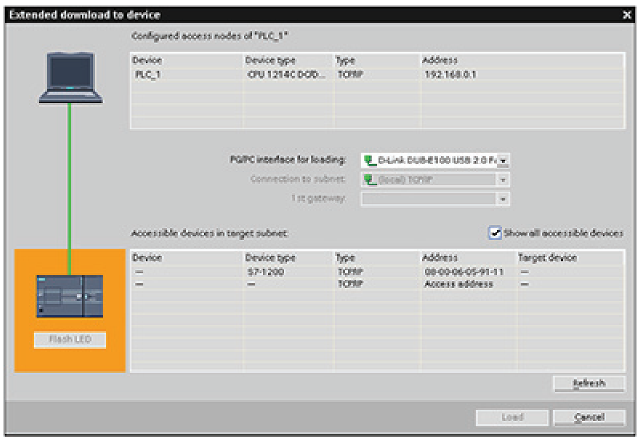
Each CPU is loaded at the factory with a permanent, unique MAC address. You cannot change the MAC address of a CPU.

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.



Extended download to device

Configured access nodes of "PLC_1"

Device	Device type	Type	Address
PLC_1	CPU 1214C DC/D...	TCP/IP	192.168.0.1

PG/PC interface for loading: D-Link DUB-E100 USB 2.0 F...
Connection to subnet: (local) TCP/IP
1 st gateway:

Accessible devices in target subnet: [✓] Show all accessible devices

Device	Device type	Type	Address	Target device
—	S7-1200	TCP/IP	08-00-06-05-91-11	—
—	—	TCP/IP	Access 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 and IP addresses. MAC addresses are all-important in identifying devices that are missing the required unique IP address.

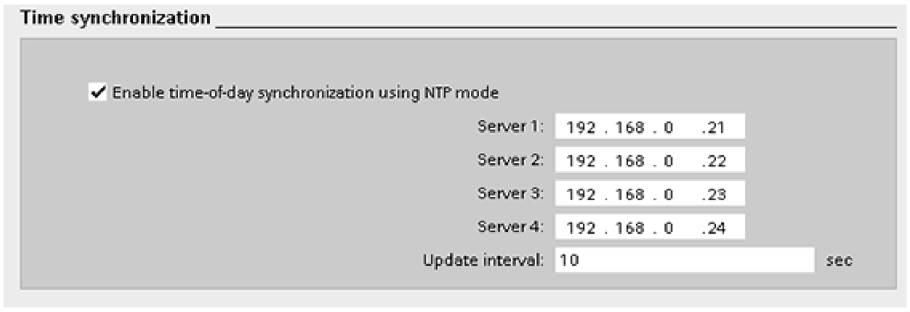
7.5.2 Configuring Network Time Protocol synchronization

The Network Time Protocol (NTP) is widely used to synchronize the clocks of computer systems to Internet time servers. It provides accuracies typically less than a millisecond on LANs and up to a few milliseconds on WANs. Typical NTP configurations utilize multiple redundant servers and diverse network paths in order to achieve high accuracy and reliability.

The NTP subnet operates with a hierarchy of levels, where each level is assigned a number called the stratum. Stratum 1 (primary) servers at the lowest level are directly synchronized to national time services. Stratum 2 (secondary) servers at the next higher level are synchronized to stratum 1 servers and so on.

Time synchronization parameters

In the Properties window, select the "Time synchronization" configuration entry. The TIA Portal displays the Time synchronization configuration dialog:


Time synchronization

✓ Enable time-of-day synchronization using NTP mode

Server 1: 192 . 168 . 0 .21
Server 2: 192 . 168 . 0 .22
Server 3: 192 . 168 . 0 .23
Server 4: 192 . 168 . 0 .24
Update interval: 10 sec

Note

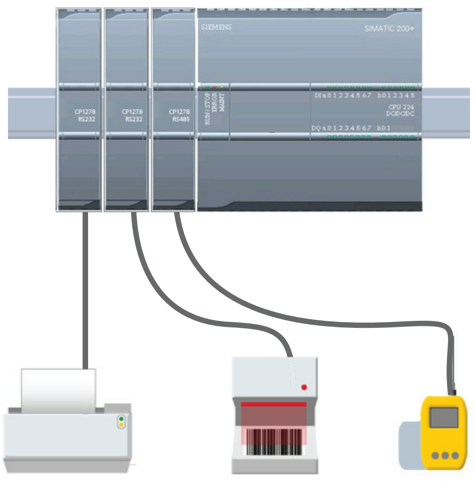
All IP addresses are configured when you download the project.

The following table defines the 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)

Point-to-Point (PtP) communications 8

The CPU supports the Point-to-Point protocol (PtP) for character-based serial communication, in which the user application completely defines and implements the protocol of choice. PtP provides maximum freedom and flexibility, but requires extensive implementation in the user program.



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 or RS485 communication module (CM) provides the electrial interface for performing the PtP communications.

STEP 7 Basic provides libraries of instructions that you can use in programming your application. These libraries provide PtP communication functions for the following protocols:

- USS drive protocol
- Modbus RTU Master Protocol
- Modbus RTU Slave Protocol

8.2 Using the RS232 and RS485 communication modules

Two communication modules (CMs) provide the interface for PtP communications: CM 1241 RS485 (Page 317) and CM 1241 RS232 (Page 318). You can connect up to three CMs (of any type). Install the CM to the left of the CPU or another CM. Refer to the "Installation" chapter (Page 29) for detailed instructions on module installation and removal.

The RS232 and RS485 communication modules have the following characteristics:

- Isolated port
- Supports Point-to-Point protocols
- Configured and programmed through extended instructions and library functions

- Displays transmit and receive activity by means of LEDs
- Displays a diagnostic LED
- Powered by the CPU. No external power connection is needed.

Refer to the Technical Specifications for Communication Modules (Page 317).

8.3 Configuring the communication ports

The communication modules can be configured by two methods:

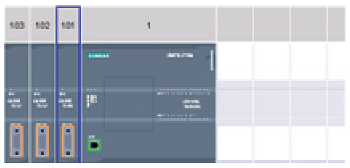
- Use the device configuration in STEP 7 Basic to configure the port parameters (baud and parity), the send parameters and the receive parameters. The device configuration settings are stored permanently in the CPU. These settings are applied after a power cycle and a RUN to STOP transition.
- Use the PORT_CFG, SEND_CFG and RCV_CFG 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.

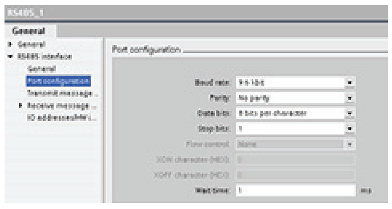
After configuring the hardware devices (Page 69), you configure parameters for the communication interfaces by selecting one of the CMs in your rack.

The "Properties" tab of the inspector window displays the parameters of the selected CM. 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, the port configuration parameters are the same regardless of whether you are configuring an RS232 or an RS485 communication module. The parameter values may differ.





Baud rate: 9.6 kbit/s
Parity: No parity
Data bits: 8 bits per character
Stop bits: 1
Flow control: None
XON character (HEX): 0
XOFF character (HEX): 0
Wait time: 1 ms

The port can also be configured (or the existing configuration can be changed) from the user program with the PORT_CFG (Page 250) instruction.

Note

Parameter values set from the PORT_CFG instruction in the user program override port configuration settings set from the STEP 7 Basic. Note that the S7-1200 does not retain parameters set from the PORT_CFG instruction in the event of power down.

Baud rate: The default value for the baud rate is 9.6 kbits per second. Valid choices are:

300 baud	2.4 kbits	19.2 kbits	76.8 kbits
600 baud	4.8 kbits	28.4 kbits	115.2 kbits
1.2 kbits	9.6 kbits	57.6 kbits	

Parity: The default value for parity is no parity. Valid choices are:

- No parity
- Even
- Odd
- Mark (parity bit always set to 1)
- 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 239)". 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 ASCII characters for the XON and XOFF characters.

The RS485 communication module does not support flow control.

Wait time: The wait time specifies the time that the communication module 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 module receives an expected CTS or XON, the communication module 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.

8.4 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 does not support flow control. You specify the type of flow control either when you configure the port (Page 238) or with the PORT_CFG instruction.

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 CM 1241 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 238).

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 node, 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. These characters are ASCII characters that represent XON and XOFF.

XOFF indicates that a transmission must stop. XON indicates that a transmission can resume.

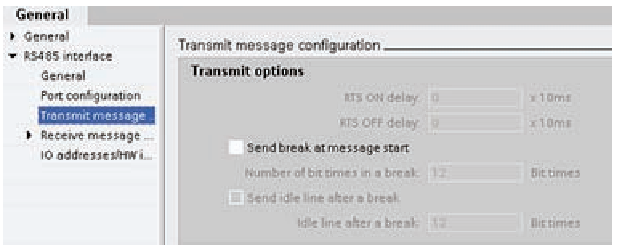
When the transmitting device receives an XOFF character from the receiving device, it stops transmitting. Transmitting resumes when the transmitting device receives an XON character. If it does not receive an XON character within the wait time that is specified in the port configuration (Page 238) , 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.

8.5 Configuring the transmit (send) and receive parameters

Before the PLC 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.

Configuring the transmit (send) parameters



Transmit message configuration

Transmit options

RTS ON delay: 0 x 10ms
RTS OFF delay: 0 x 10ms
Send break at message start
Number of bit times in a break: 12 Bit times
Send idle line after a break
Idle line after a break: 12 Bit times

During the configuration of the CM, you configure how a communication interface transmits data by specifying the "Transmit message configuration" property for the selected CM.

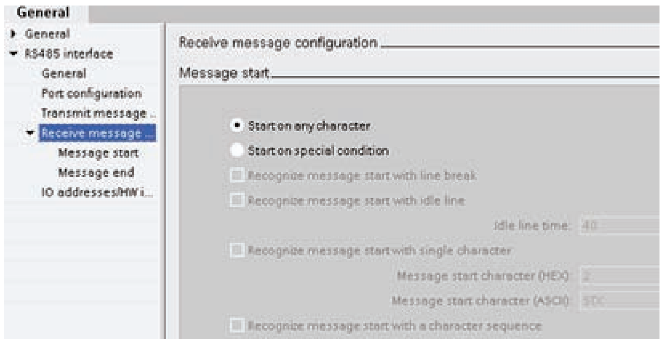
You can also dynamically configure or change the transmit message parameters from the user program by using the SEND_CFG (Page 251) instruction.

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 238) 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 238) specifies hardware flow control. This parameter is applicable for RS232 modules only.
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 after a break at message start. 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.

Configuring the Receive parameters



Receive message configuration

Message start

● Start on any character
○ Start on special condition
☐ Recognize message start with line break
☐ Recognize message start with idle line
Idle line time: 40
☐ Recognize message start with single character
Message start character (HEX): 2
Message start character (ASCII): STX
☐ Recognize message start with a character sequence

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 CM.

You can also dynamically configure or change the receive message parameters from the user program by using the RCV_CFG (Page 253) instruction.

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.

For more information, see the RCV_CFG instruction.

Message start parameters

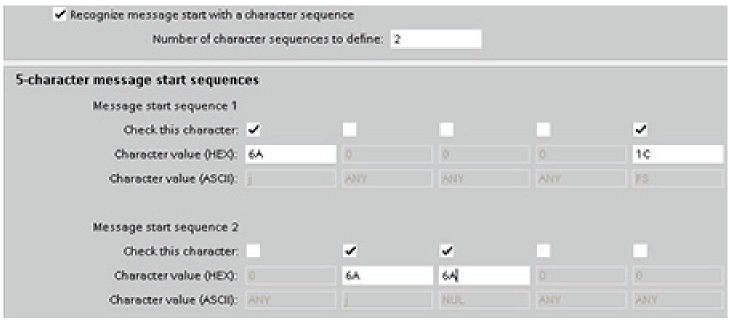
You can determine how the communication module recognizes the start of a message. The start characters and characters comprising the message go into the receive buffer until a configured end condition is met.

Multiple start conditions can be specified. 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 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 will restart the start of message search by again looking for an idle line time.

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 will restart the checking with the first required condition.

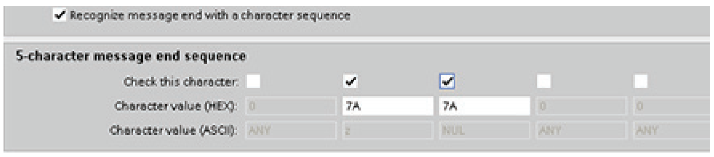
Parameter	Definition
Start Character character	The Start Character condition specifies that successfully receiving a particular character will begin a message. This character will be the first character within a message. Any character that is received before this specific character will be discarded.
Start on Any Character	The Any Character condition specifies any successfully received character will begin the start of a message. This character will be the first character within a message.
Line Break	The Line Break conditions specifies that a message receive operation should start after a break character is received.
Idle Line	The Idle Line condition specifies that a message reception should start 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 will begin.
Special condition: Recognize message start with single	Specifies that a particular character indicates the start of a message. The default is STX.
Special condition: Recognize message start with a character sequence	Specifies that a particular character sequence 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. Incoming sequences will be 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 five specific character sequences, which you can enable or disable as needed. The start condition is satisfied when any one of the configured character sequences occurs.
Sample configuration	[✓] Recognize message start with a character sequence Number of character sequences to define: 2 **5-character message start sequences** Message start sequence 1 Check this character: ✓, ☐, ☐, ☐, ✓ Character value (HEX): 6A, 0, 0, 0, 1C Character value (ASCII): j, ANY, ANY, ANY, FS Message start sequence 2 Check this character: ☐, ✓, ✓, ☐, ☐ Character value (HEX): 0, 6A, 6A, 0, 0 Character value (ASCII): ANY, j, NUL, ANY, ANY 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: • <any character> 6A 6A • 6A 12 14 18 1C • 6A 44 A5 D2 1C

Message end parameters

You can 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.

Multiple end conditions can be specified at the same time. The message will end when any one of the end conditions has been satisfied. 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 message end has expired. The message timeout period begins when the first character is received according to the message start criteria. The default is 200 ms and the range is 0 to 65535 ms.
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. The default response timeout is 200 ms and the range is 0 to 65535 ms. You must configure another end condition to indicate the actual end of a message.
Recognize message end by inter-character gap	The message end occurs when the maximum configured timeout between 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.
Recognize message end by max length	The message end occurs when the configured maximum number of characters has been received. The default is 0 bytes and the maximum is 1024 bytes.
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.

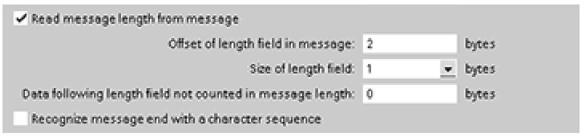
Parameter	Definition
Sample configuration	 ✓ Recognize message end with a character sequence **5-character message end sequence** Check this character: ☐ ✓ ✓ ☐ ☐ Character value (HEX): 0 7A 7A 0 0 Character value (ASCII): ANY z NUL ANY ANY In this case, the end condition is satisfied when two consecutive 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 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


✓ Read message length from message
Offset of length field in message: 2 bytes
Size of length field: 1 bytes
Data following length field not counted in message length: 0 bytes
☐ Recognize message end with a character sequence

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).

8.6 Programming the PtP communications

STEP 7 Basic 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 communication module through the device configuration or through these instructions in your user program:

- PORT_CFG
- SEND_CFG
- RCV_CFG

Communication instructions

The PtP communication instructions enable the user program to send messages to and receive messages from the communication modules. For information transferring data with these instructions, see the section on data consistency (Page 86).

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 buffer the transmit and receive messages as necessary up to a maximum buffer size of 1024 bytes.

The communication modules 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.	• SEND_PTP • RCV_PTP
Additional instructions provide the capability to reset the receive buffer, and to get and set specific RS232 signals.	• RCV_RST • SGN_GET • SGN_SET

8.6.1 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 communication module.
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 has collected a response message, the RCV_PTP instruction will copy the response to the CPU and indicate 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 should execute the RCV_PTP instruction every scan.
2. When the CM has received a request, the RCV_PTP instruction will indicate that new data is ready and the request will be copied into the CPU.
3. The user program should service the request and generate 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 should receive transmissions without missing any.

8.7 Point-to-Point instructions

8.7.1 Common parameters for Point-to-Point instructions

Communication module LED behaviors

There are three LED indicators on the Communication module (CM):

- Diagnostic LED: This LED flashes red until it is addressed by the CPU. After the CPU powers up, it will check for modules and address the CM module. The Diagnostic LED will begin to flash green. This means that the CPU has addressed the CM, but has not yet provided configuration to the CM. The configuration is downloaded to the module when the program is downloaded to the CPU. After a download to the CPU, the Diagnostic LED on the communication module should be a steady green.
- Transmit LED: This LED is located above the receive LED. The transmit LED illuminates when data is being transmitted out the communication port.
- Receive LED: This LED illuminates when data is being received by the communication port.

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 of amount of time that can be measured by a S7-1200 is 8 seconds.

REQ input parameter

Many of the Point-to-Point (PtP) instructions use a REQ input that initiates 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 will not initiate another operation until it is 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, you are prompted to identify the instance DB. Use a unique DB for each type of PtP instruction. That is, all SEND_PTP instructions for a given port should have the same instance DB, but SEND_PTP and RCV_PTP must have different instance DBs. This ensures that inputs such as the REQ are properly handled by each instruction.

PORT input parameter

Select from the drop down menu (associated with the PORT input) the port identifier for the CM that you want this instance of the instruction to operate. This number is also found as the "hardware identifier" in the configuration information for the CM.

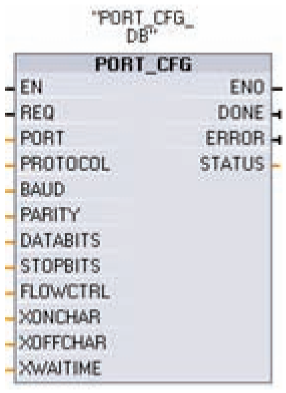
NDR, DONE, ERROR, and STATUS output parameters

- The output DONE indicates that the requested operation has completed without error. This output will be set for one scan.
- The output NDR (New Data Ready) indicates that the requested action has completed without error and new data has been received. This output will be set for one scan.
- The output ERROR indicates that the requested action has completed with an error. This output will be set for one scan.
- The output STATUS is used to report errors or intermediate status results.
  - If the DONE or NDR bit is set, then STATUS will be set to 0 or to an informational code.
  - If the ERROR bit is set, then STATUS will be 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, such as a busy status.

Common condition codes

STATUS (W#16#....)	Description
0000	No error
8x3A	Illegal pointer in parameter x
8070	All internal instance memory in use
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 returned a received message with a length greater than the length parameter allowed.
8090	Wrong message length, wrong submodule, or illegal message
8091	Wrong version in parameterization message
8092	Wrong record length in parameterization message

8.7.2 PORT_CFG instruction



PORT_CFG (Port Configuration) 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. 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.

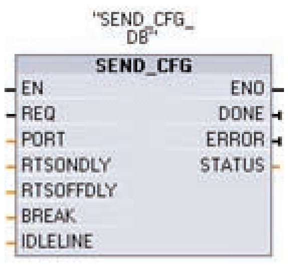
See Configuring the communication ports (Page 238) and Managing flow control (Page 239) for more information.

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Activate the configuration change on rising edge of this input.
PORT	IN	PORT	Communication port identifier: This logical address is a constant which can be referenced within the "Constants" tab of the default tag table.
PROTOCOL	IN	UInt	0 - Point-to-Point communication protocol 1..n - future definition for specific protocols
BAUD	IN	UInt	Port baud rate: 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: 1 - No parity 2 - Even parity 3 - Odd parity 4 - Mark parity 5 - Space parity
DATABITS	IN	UInt	Bits per character: 1 - 8 data bits 2 - 7 data bits
STOPBITS	IN	UInt	Stop bits: 1 - 1 stop bit 2 - 2 stop bits

Parameter	Parameter type	Data type	Description
FLOWCTRL	IN	UInt	Flow control: 1 - No flow control 2 - XON/XOFF 3 - Hardware RTS always ON 4 - Hardware RTS switched
XONCHAR	IN	Char	Specify 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.
XOFFCHAR	IN	Char	Specify 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.
XWAITIME	IN	UInt	Specify how long to wait for a XON character after receiving a XOFF character, or how long to wait for the CTS signal after enabling RTC (0 to 65535 ms). This parameter is only evaluated if flow control is enabled.
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

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

8.7.3 SEND_CFG instruction



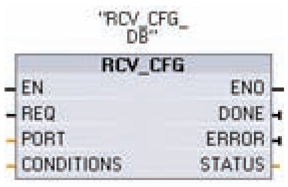
SEND_CFG (Send Configuration) allows the dynamic configuration of serial transmission parameters for a Point-to-Point communication port. Any queued messages within a communication module (CM) will be discarded once SEND_CFG is executed.

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 PLC. 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 transmit (send) and receive parameters (Page 241).

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Activate the configuration change on the rising edge of this input.
PORT	IN	PORT	Communication port identifier: This logical address is a constant which can be referenced within the "Constants" tab of the default tag table.
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. 0 - 65535 ms. 0 will disable the feature.
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. 0 - 65535 ms. 0 will disable the feature.
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. 0 will disable the feature. 8 second maximum
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. 0 will disable the feature. 8 second maximum
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

STATUS (W#16#....)	Description
80B0	Transmit interrupt configuration is not allowed
80B1	Break time is greater than the allowed value (2500 bit times)
80B2	Idle time is greater than the allowed value (2500 bit times)

8.7.4 RCV_CFG instruction



RCV_CFG (Receive Configuration) performs dynamic configuration of serial receiver parameters for a Point-to-Point communication port. This instruction configures the conditions that signal the start and end of a received message. Any queued messages within a CM will be discarded when RCV_CFG is executed.

You can set up the initial static configuration of the CM 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 PLC. 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 Receive parameters (Page 241) for more information.

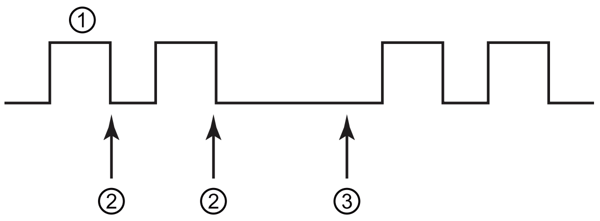
Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Activate the configuration change on the rising edge of this input.
PORT	IN	PORT	Communication port identifier: This logical address is a constant which can be referenced within the "Constants" tab of the default tag table.
CONDITIONS	IN	CONDITIONS	The Conditions data structure specifies the starting and ending message conditions. These are 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

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. Possible start conditions:

- Start Character specifies that successfully receiving a particular character will begin a message. This character will be the first character within a message. Any character that is received before this specific character will be discarded.
- Any Character specifies that any successfully received character will begin the start of a message. This character will be the first character within a message.
- Line Break specifies that a message receive operation should start after a break character is received.

- Idle Line specifies that a message reception should start 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 will begin.



① Characters

② Restarts the idle line timer

③ Idle line is detected and message receive is started

- Variable Sequences: Start conditions can be constructed that are based upon a variable number of character sequences (up to a maximum of 4) consisting of a varying number of characters (up to a maximum of 5). Each character position within each sequence may be selected as a specific character, or selected as a wild-card character, meaning any character will qualify. This start condition can be used when different sequences of characters indicate the start of a message.

  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.

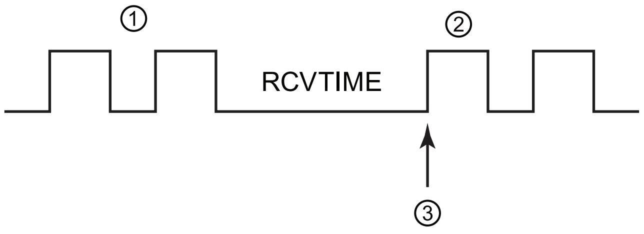
  The start sequence processing can be terminated due to various parity, framing, or inter-character timing errors. These errors result in no received message, because the start condition was not satisfied.

Start condition	First Character	First Character +1	First Character +2	First Character +3	First Character +4
1	**68H**	xx	xx	**68H**	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. Possible message end conditions:

- Response Timeout specifies that a character of the response should be successfully received within the time specified by RCVTIME. The timer begins as soon as the transmission completes successfully and the module begins the receive operation. If a character is not received within the RCVTIME period, 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 should be successfully received within the specified time. A distinct end condition must be used to define the end condition for the response messages.

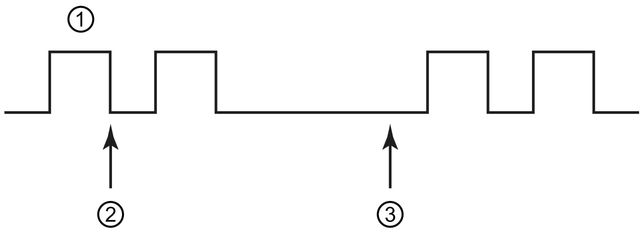


① Transmitted characters

② Received characters

③ The first character must be successfully received by this time

- Message Timeout specifies that a message should be received within the time specified by MSGTIME. The timer begins as soon as the specified start condition has been satisfied.

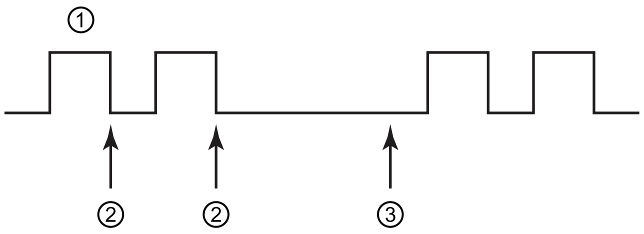


① Received characters

② Start Message condition satisfied: message timer starts

③ Message timer expires and terminates the message

- Intercharacter Gap is the time measured from the end of one character (the last stop bit) until the end of the next character. If the time between any two characters exceeds the number of configured bit times, the message will be terminated.



①	Received characters
②	Restarts the intercharacter timer
③	The intercharacter timer expires and terminates the message with errors

- Maximum Length: The receive operation will stop once the specified number of characters have been received. 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.

- Combination Condition of "N + Length Size + Length M". This end condition can be used to process a varying sized message that contains a length field.
  - N specifies the position (number of characters into the message) where the length field begins. (1 based)
  - Length Size specifies the size of the length field. Valid values are 1, 2, or 4 bytes.
  - Length M specifies the number of ending characters (following the length field) that are not included within the length of the message. This value can be used to specify the length of a checksum field whose size is not included in the length field.
  - As an example, consider a message format that consists of a start character, an address character, a one-byte length field, message data, checksum characters, and an end character. The entries identified with "Len" correspond with the N parameter. The value of N would be 3, specifying that the length byte is positioned 3 bytes into the message. The value of Length Size would be 1, specifying that the value for the length of the message is contained in 1 byte. The checksum and end char fields correspond with the "Length M" parameter. The value of "Length M" would be 3, specifying the number of bytes of the checksum and character fields.

Start char (1)	**Address** (2)	**Len** (N) (3)	**Message** ... (x)		**Checksum and End char Length M** x+1 x+2 x+3		
xx	xx	xx	xx	xx	xx	xx	xx

- Variable Characters: This end condition can be used to end receiving based upon different character sequences. The sequences can consist of a varying number of characters (up to a maximum of 5). Each character position within each sequence can be selected as a specific character, or selected as a wild-card character, meaning any character will satisfy the condition. Any leading characters that are configured to be ignored are not required to be part of the message. Any trailing characters that are ignored are required to be part of the message.

Parameter CONDITIONS data type structure part 1 (start conditions)

Parameter	Parameter type	Data type	Description
STARTCOND	IN	UInt	Specify the start condition: • 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. Only used with an idle line condition. 0 to 65535
STARTCHAR	IN	Byte	The start character used with the start character condition.
STRSEQ1CTL	IN	Byte	Sequence 1 ignore/compare control for each character: 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)
STRSEQ2CTL	IN	Byte	Sequence 2 ignore/compare control for each character
STRSEQ2	IN	Char[5]	Sequence 2 start characters (5 characters)
STRSEQ3CTL	IN	Byte	Sequence 3 ignore/compare control for each character
STRSEQ3	IN	Char[5]	Sequence 3 start characters (5 characters)
STRSEQ4CTL	IN	Byte	Sequence 4 ignore/compare control for each character
STRSEQ4	IN	Char[5]	Sequence 4 start characters (5 characters)

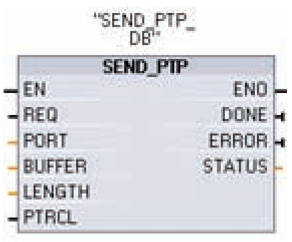
Parameter CONDITIONS data type structure part 2 (end conditions)

Parameter	Parameter type	Data type	Description
ENDCOND	IN	UInt	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	Maximum message length: Only used when the maximum length end condition is selected. 0 to 1023 bytes
N	IN	UInt	Byte position within the message of the length field. Only used with the N + LEN + M end condition. 1 to 1023 bytes
LENGTHSIZE	IN	UInt	Size of the byte field (1, 2, or 4 bytes). Only used with the N + LEN + M end condition.
LENGTHM	IN	UInt	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	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 time, 8 second maximum This parameter is not really evaluated as an end condition since it only evaluates start conditions. A distinct end condition must be selected.
MSGTIME	IN	UInt	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 - 65535 milliseconds
CHARGAP	IN	UInt	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 milliseconds
ENDSEQ1CTL	IN	Byte	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]	Sequence 1 start characters (5 characters)

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	Max 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 or > 2500
80CA	Idle line timeout is enabled and it is zero or > 2500
80CB	End sequence is enabled but all chars are "don't care"
80CC	Start sequence (any one of 4) is enabled but all chars are "don't care"

8.7.5 SEND_PTP instruction



SEND_PTP (Send Point-to-Point data) initiates the transmission of the data. SEND_PTP transfers the specified buffer to the CM. The CPU program continues while the CM sends the data at the specified baud rate. Only one send operation can be pending at a given time. The CM returns an error if a second SEND_PTP is executed while the CM is already transmitting a message.

Parameter	Parameter 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 module (CM).
PORT	IN	PORT	Communication port identifier: This logical address is a constant which can be referenced within the "Constants" tab of the default tag table.
BUFFER	IN	Variant	This parameter points to the starting location of the transmit buffer. Boolean data or Boolean arrays are not supported.
LENGTH	IN	UInt	Transmitted frame length from bytes When transmitting a complex structure, always use a length of 0.

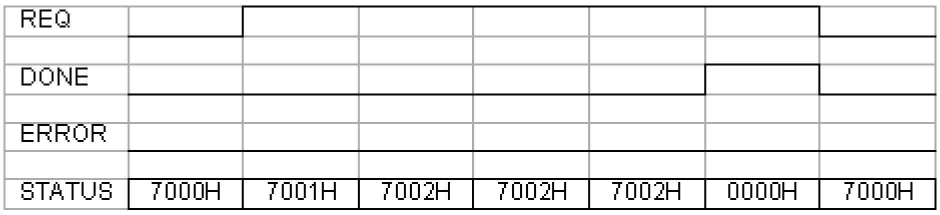
Parameter	Parameter type	Data type	Description
PTRCL	IN	Bool	This parameter selects the buffer as normal point-to-point or specific Siemens-provided protocols that are implemented within the attached CM. FALSE = user program controlled point-to-point operations. (only valid option)
DONE	OUT	Bool	TRUE 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

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 for one scan cycle to show the status of the transmit operation. While DONE or ERROR is TRUE, the STATUS output is valid.

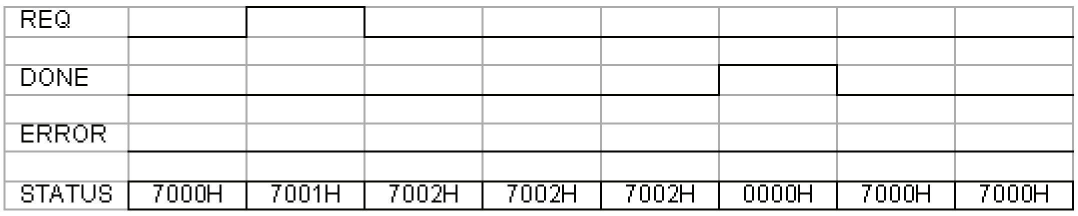
The instruction returns a status of 16#7001 if the communication module (CM) accepts the transmit data. Subsequent SEND_PTP executions return a 16#7002 if the CM is still busy transmitting. When the transmit operation is complete, the CM will return the status of the transmit operation, 16#0000, if no errors occurred. Subsequent executions of SEND_PTP with REQ low return a status of 16#7000 (not busy).

Relationship of the output values to REQ:

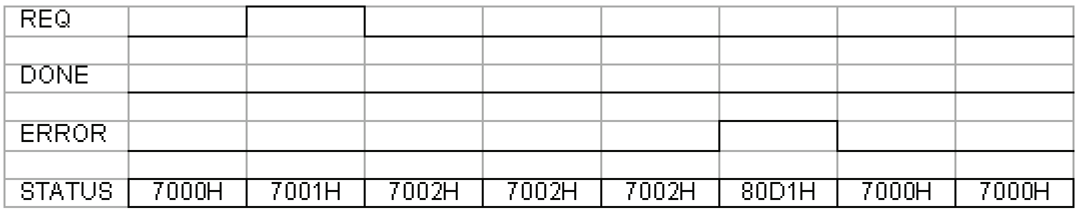
This assumes that the instruction is called periodically to check for the status of the transmission process. In the diagram below, it is assumed that the instruction is called every scan (represented by the STATUS values).



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.



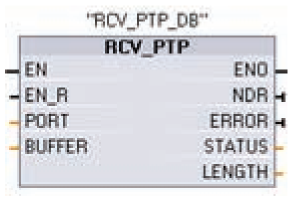
STATUS (W#16#....)	Description
80D0	New request while transmitter active
80D1	Transmit aborted because of no CTS within wait time
80D2	Transmit aborted because of no DSR from the DCE device
80D3	Transmit aborted because of queue overflow (transmit more than 1024 bytes)
7000	Not busy
7001	Busy when accepting request (first call)
7002	Busy on a poll (nth call)

Interaction of the LENGTH and DATA parameters for PTP_SEND

The minimum size of data that can be transmitted by the PTP_SEND instruction is a byte. The DATA parameter determines the size of the data to be transmitted. You cannot use BOOL or arrays of BOOL for the DATA parameter.

LENGTH parameter	DATA parameter	Description
LENGTH = 0	Not used	The complete data is sent as defined at the DATA parameter. You do not need to specify the number of transmitted bytes when LENGTH = 0.
LENGTH > 0	Elementary data type	The LENGTH value must contain the byte count of this data type. Otherwise, nothing is transferred and the error 8088H is returned.
	Structure	The LENGTH value may contain a byte count less than the complete byte length of the structure. In this case, only the first LENGTH bytes are transferred.
	Array	The LENGTH value may contain a byte count that is less than the complete byte length of the array. In this case, only array elements which fit completely in LENGTH bytes are transferred. The LENGTH value must be a multiple of the data element byte count. Otherwise, STATUS = 8088H, ERROR = 1, and no transmit occurs.
	String	The complete memory layout of the string format is transferred. The LENGTH value must include bytes for maximum length, actual length, and the string characters. For the STRING data type, all lengths and characters have a byte size. If a string is used as actual parameter at the DATA parameter, the LENGTH value must also include two bytes for the two length fields.

8.7.6 RCV_PTP instruction



RCV_PTP (Receive Point-to-Point) checks for messages that have been received in the CM. If a message is available, it will be transferred from the CM to the CPU. An error returns the appropriate STATUS value.

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. This will typically be 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), that indicates the receive operation was terminated for an error condition such as parity, framing, or overrun errors.

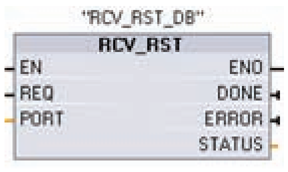
Each Point-to-Point CM module can buffer up to a maximum of 1K bytes. This could be one large message or several smaller messages.

Parameter	Parameter type	Data type	Description
EN_R	IN	Bool	When this input is TRUE, check the CM module for received messages. If a message was successfully received, it will be transferred from the module to the CPU. When EN_R is FALSE, the CM is checked for received messages and the STATUS output is set, but the message is not transferred to the CPU.
PORT	IN	PORT	Communication port identifier: This logical address is a constant that can be referenced within the "Constants" tab of the default tag table.
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.
NDR	OUT	Bool	TRUE for one scan, when new data is ready and 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
LENGTH	OUT	UInt	Length of the returned message (in bytes)

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

STATUS (W#16#...)	Description
80E4	Message terminated because calculated length exceeds buffer size
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
0099	Message terminated because of end sequence was satisfied

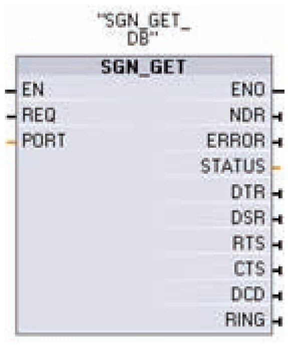
8.7.7 RCV_RST instruction



RCV_RST (Receiver Reset) clears the receive buffers in the CM.

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Activates the receiver reset on the rising edge of this enable input
PORT	IN	PORT	Communication port identifier: The port must be specified using the module’s logical address.
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

8.7.8 SGN_GET instruction

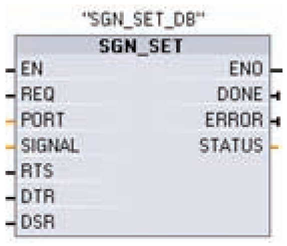


SGN_GET (Get RS232 Signals) reads the current states of RS232 communication signals. This function is only valid for the RS232 CM (communication module).

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Get RS232 signal state values on the rising edge of this input
PORT	IN	PORT	Communication port identifier: This logical address is a constant which can be referenced within the "Constants" tab of the default 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
DTR	OUT	Bool	Data terminal ready, module ready (output)
DSR	OUT	Bool	Data set ready, communication partner ready (input)
RTS	OUT	Bool	Request to send, module ready to send (output)
CTS	OUT	Bool	Clear to send, communication partner can receive data (input)
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)

STATUS (W#16#....)	Description
80F0	CM is RS485 module and no signals are available
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

8.7.9 SGN_SET instruction



SGN_SET (Set RS232 Signals) sets the states of RS232 communication signals. This function is only valid for the RS232 CM (communication module).

Parameter	Parameter type	Data type	Description
REQ	IN	Bool	Start the set RS232 signals operation, on the rising edge of this input
PORT	IN	PORT	Communication port identifier: This logical address is a constant that can be referenced within the "Constants" tab of the default tag table.
SIGNAL	IN	Byte	Selects which signal to set: (multiple allowed) • 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)
DTR	IN	Bool	Data terminal ready, module ready to send value to set (true or false)
DSR	IN	Bool	Data set ready (only applies to DCE type interfaces) (not used)
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

STATUS (W#16#....)	Description
80F0	CM is RS485 module and no signals are settable
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

8.8 Errors

Return values of PtP instructions

Each PtP instruction has three outputs that provide the completion status:

Parameter	Data type	Default	Description
DONE	Boolean	FALSE	TRUE for one scan indicates that the last request completed without errors.
ERROR	Boolean	FALSE	TRUE indicates that the last request completed with errors, with the applicable error code in STATUS.
STATUS	Word	0	Two bytes that contain the error class and error number, if applicable. STATUS retains its value for the duration of the execution of the function.

Common error classes and errors

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

Port configuration errors

Event / error ID	Description
0x80A0	The specific protocol does not exist
0x80A1	The specific baud rate does not exist
0x80A2	The specific parity does not exist
0x80A3	The specific number of data bits does not exist
0x80A4	The specific number of stop bits does not exist
0x80A5	The specific type of flow control does not exist

Transmit configuration errors

Event / error ID	Description
0x80B0	The specific protocol does not exist
0x80B1	The specific baud rate does not exist
0x80B2	The specific parity does not exist
0x80B3	The specific number of data bits does not exist
0x80B4	The specific number of stop bits does not exist
0x80B5	The specific type of flow control does not exist

Receive configuration errors

Event / error ID	Description
0x80C0	Start condition error
0x80C1	End condition error
0x80C3	Maximum length error
0x80C4	N value error (refer to N+LEN+M)
0x80C5	Length size error (refer to MAXLEN or N+LEN+M)
0x80C6	M value error (refer to N+LEN+M)
0x80C7	N-Length-M value error (refer to N+LEN+M)
0x80C8	Response timeout error, no message was received during the specified receive period. (refer to RCVTIME or MSGTIME)
0x80C9	Inter-character timeout error (refer to CHARGAP)
0x80CA	Idle line timeout error (refer to Idle Line)
0x80CB	A specified end sequence is configured with all "don't care" characters
0x80CC	A specified start sequence is configured with all "don't care" characters

Signal errors

Event / error ID	Description
0x80F0	The communication module is an RS485 module and no signals are available
0x80F1	The communication module is an RS232 module, but no signals are settable because H/W flow control is enabled
0x80F2	The DSR signal can not be set since the module is a DTE device

Transmission runtime errors

Event / error ID	Description
Buffer Limit	The total available transmit buffer of the CP has been exceeded
0x80D0	A new request was received while the transmitter was active

Event / error ID	Description
0x80D1	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
0x80D2	The transmit request was aborted because no DSR signal is received from the DCE
0x80D3	The total available transmit buffer of the CP has been exceeded
0x7000	The transmit function is not busy
0x7001	The transmit function is busy with the first call
0x7002	The transmit function is busy with subsequent calls (polls after the first call)

Reception runtime return values

Event / error ID	Description
0x80E0	The message was terminated because the receive buffer is full
0x80E1	The message was terminated as a result of a parity error
0x80E2	The message was terminated as a result of a framing error
0x80E3	The message was terminated as a result of an overrun error
0x80E4	The message was terminated as a result of the specified length exceeding the total buffer size
0x0094	The message was terminated because the maximum character length was received (MAXLEN)
0x0095	The message was terminated because the complete message was not received in the specified time (MSGTIME)
0x0096	The message was terminated because the next character was not received in the within the duration of the inter-character time (CHARGAP)
0x0097	The message was terminated because the first character was not received in the specified time (RCVTIME)
0x0098	The message was terminated because the "n+len+m" length condition has been satisfied (N+LEN+M)
0x0099	The message was terminated because the end sequence has been satisfied (ENDSEQ)

Miscellaneous parameter errors

Event / error ID	Description
0x8n3A	An illegal pointer was provided on parameter n
0x8070	All internal instance memory is in use
0x8080	The port number is invalid
0x8082	Parameterization failed because parameterization is already in progress in the background
0x8083	Buffer overflow. CM returned more data than allowed.
0x8085	LEN parameter has the value of 0 or is greater than the largest value allowed
0x8088	LEN parameter is larger than the memory area specified in DATA

Online and diagnostic tools 9

9.1 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. The CPU provides the following status indicators:

- STOP/RUN
  - Solid orange indicates STOP mode
  - Solid green indicates RUN mode
  - Flashing (alternating green and orange) indicates that the CPU is starting up
- 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

Description	STOP/RUN Orange / Green	ERROR Red	MAINT Orange
Power is off	Off	Off	Off
Startup, self-test, firmware update	Flashing (alternating orange and green)	-	Off
Stop mode	On (orange)	-	-
Run mode	On (orange)	-	-
Remove the memory card	On (orange)	-	Flashing
Error	On (either orange or green)	Flashing	-
Maintenance requested	On (either orange or green)	-	On
Defective hardware	On (orange)	On	Off
LED test or defective CPU firmware	Flashing (alternating orange 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.

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).

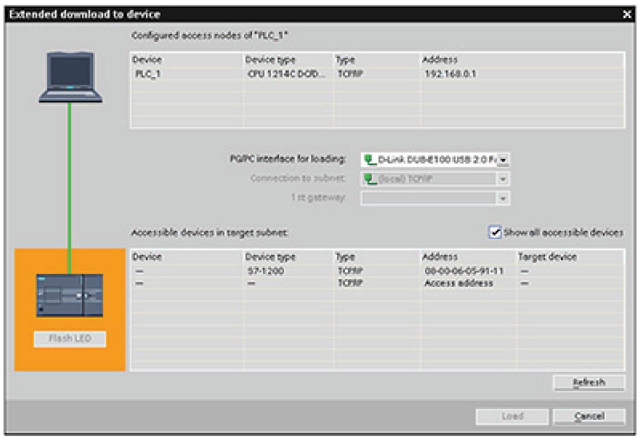
Description	DIAG (Red / Green)	I/O Channel (Red / Green)
Field-side power is off	Flashing red	Flashing red
Not configured or updated in progress	Flashing green	Off
Module configured with no errors	On (green)	On (green)
Error condition	Flashing red	-
I/O error (with diagnostics enabled)	-	Flashing red
I/O error (with diagnostics disabled)	-	On (green)

9.2 Going online and connecting to a CPU

An online connection between the programming device and a target system is required for loading programs and project engineering data to the target system as well as for activities such as the following:

- Testing user programs
- Displaying and changing the operating mode of the CPU
- Displaying and setting the date and time of day of the CPU
- Displaying the module information
- Comparing online and offline blocks
- Diagnosing hardware

You can then access the data on the target system in the online or diagnostics view using the "Online tools" task card.


Extended download to device

Configured access nodes of "PLC_1"

Device	Device type	Type	Address
PLC_1	CPU 1214C DC/D...	TCP/IP	192.168.0.1

PG/PC interface for loading: D-Link DUB-E100 USB 2.0 F...
Connection to subnet: (local) TCP/IP
1st gateway:

Accessible devices in target subnet: ✓ Show all accessible devices

Device	Device type	Type	Address	Target device
—	S7-1200	TCP/IP	08-00-06-05-91-11	—
—	—	TCP/IP	Access address	—

The current online status of a device is indicated by an icon to the right next to the device in the project navigation.

The orange color indicates an online connection.

Select "Accessible Nodes" to find a CPU on the network.

Go online Click "Go online" to connect to a CPU on the network.

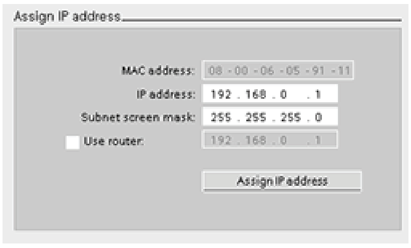
9.3 Setting the IP address and time of day

You can set the IP address and time of day in the online CPU.

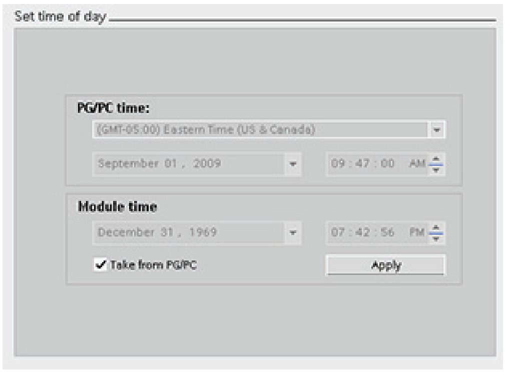
After connecting to an online CPU from the "Online & diagnostics" area, you can display or change the IP address.

Refer to the section on the IP address (Page 76) for more information.

You can also display or set the time and date parameters of the online CPU.


Assign IP address

MAC address: 08 - 00 - 06 - 05 - 91 - 11
IP address: 192 . 168 . 0 . 1
Subnet screen mask: 255 . 255 . 255 . 0
Use router: 192 . 168 . 0 . 1


Set time of day

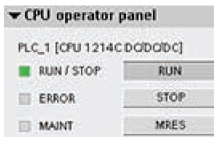
PG/PC time:
(GMT-05:00) Eastern Time (US & Canada)
September 01 , 2009 09 : 47 : 00 AM

Module time
December 31 , 1969 07 : 42 : 56 PM
✓ Take from PG/PC

https://sites.google.com/site/chauchiduc

9.4 CPU operator panel for the online CPU

The "CPU operator panel" task card 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 to change the operating mode of an online CPU.

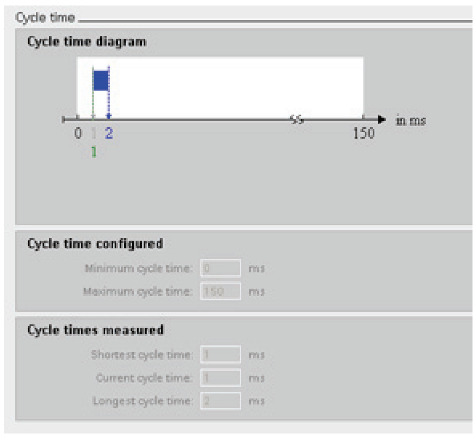


9.5 Monitoring the cycle time and memory usage

You can monitor the cycle time and memory usage of an online CPU.

After connecting to the online CPU, you can view the following measurements:

- Cycle time
- Memory usage



Cycle time configured

Minimum cycle time: 0 ms

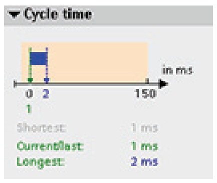
Maximum cycle time: 150 ms

Cycle times measured

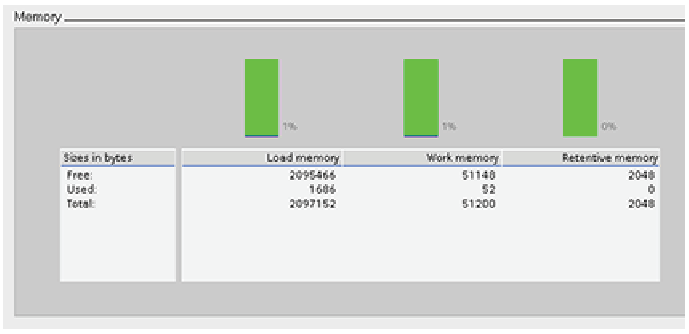
Shortest cycle time: 1 ms

Current cycle time: 1 ms

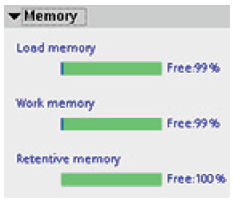
Longest cycle time: 2 ms



Shortest:	1 ms
Current/last:	1 ms
Longest:	2 ms



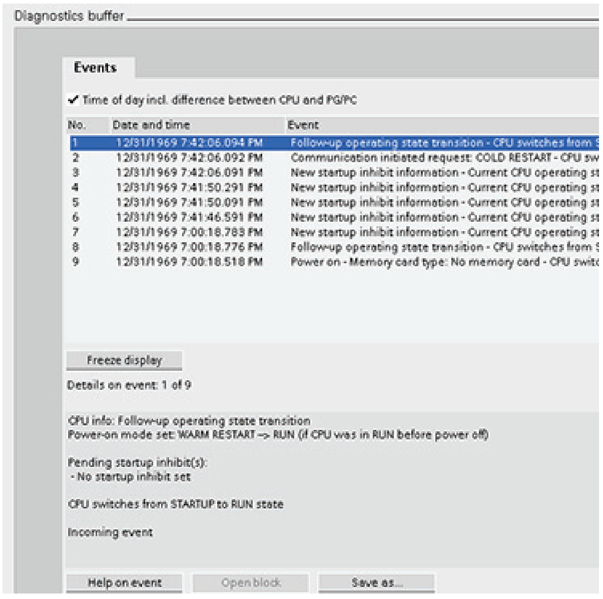
Sizes in bytes	Load memory	Work memory	Retentive memory
Free:	2095466	51148	2048
Used:	1686	52	0
Total:	2097152	51200	2048



9.6 Displaying diagnostic events in the CPU

Use the diagnostics buffer to review the recent activity in the CPU. The diagnostics buffer contains the following entries:

- Diagnostic events
- Changes in the CPU operating mode (transitions to STOP or RUN mode)



✓ Time of day incl. difference between CPU and PG/PC

No.	Date and time	Event
1	12/31/1969 7:42:06.094 PM	Follow-up operating state transition - CPU switches from S
2	12/31/1969 7:42:06.092 PM	Communication initiated request: COLD RESTART - CPU sw
3	12/31/1969 7:42:06.091 PM	New startup inhibit information - Current CPU operating st
4	12/31/1969 7:41:50.291 PM	New startup inhibit information - Current CPU operating st
5	12/31/1969 7:41:50.091 PM	New startup inhibit information - Current CPU operating st
6	12/31/1969 7:41:46.591 PM	New startup inhibit information - Current CPU operating st
7	12/31/1969 7:00:18.783 PM	New startup inhibit information - Current CPU operating st
8	12/31/1969 7:00:18.776 PM	Follow-up operating state transition - CPU switches from S
9	12/31/1969 7:00:18.518 PM	Power on - Memory card type: No memory card - CPU switc

Details on event: 1 of 9

```
CPU info: Follow-up operating state transition
Power-on mode set: WARM RESTART --> RUN (if CPU was in RUN before power off)

Pending startup inhibit(s):
 - No startup inhibit set

CPU switches from STARTUP to RUN state

Incoming event
```

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.

9.7 Watch tables for monitoring the user program

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), physical (I_:P or Q_:P), M, or DB depending on the monitor or control function.

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.

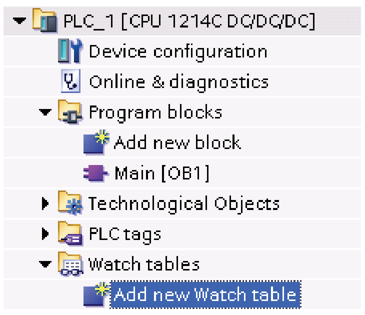
Control functions enable the user to control the sequence and the data of the program. Caution must be exercised when using control functions. These functions can seriously influence the execution of the user/system program. The three control functions are Modify, Force and Enable Outputs in STOP.

With the watch table, you can perform the following online functions:

- Monitoring the status of the tags
- Modifying values for the individual tags
- Forcing a tag to a specific value

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:

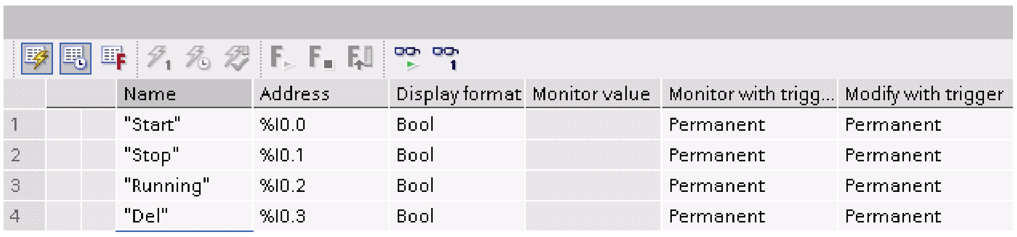
- Monitor all: This command starts the monitoring of the visible tags in the active watch table.
- Monitor now: This command starts the monitoring of the visible tags in the active watch table. The watch table monitors the tags immediately and once only.

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 trigg...	Modify with trigger
1		"Start"	%I0.0	Bool		Permanent	Permanent
2		"Stop"	%I0.1	Bool		Permanent	Permanent
3		"Running"	%I0.2	Bool		Permanent	Permanent
4		"Del"	%I0.3	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.

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.

Trigger Type	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.

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.
- 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.

Forcing values in the CPU

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 starting force.

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.

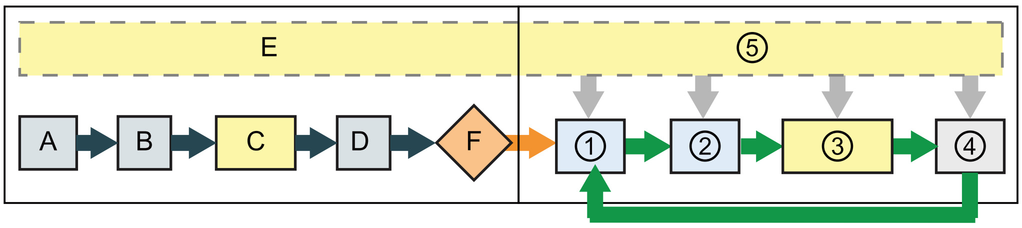
When an input or output is forced in the watch table, the force actions become part of the user program. Even though the programming software has been closed, the force selections remain active in the operating CPU program until they are cleared by going online with the programming software and stopping 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.

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

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)

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)

Industrial environments

The S7-1200 automation system is designed for use in 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.

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 electromagnetic field	80 to 1000 MHz, 10 V/m, 80% AM at 1 kHz 1-4 to 2.0 GHz, 3 V/m, 80% AM a 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.

Electromagnetic Compatibility - Immunity per EN 61000-6-2	
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

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 0.5 MHz to 5 MHz 5 MHz to 30 MHz	 <79dB (μV) quasi-peak; <66 dB (μV) average <73dB (μV) quasi-peak; <60 dB (μV) average <73dB (μV) quasi-peak; <60 dB (μV) average
Radiated Emissions EN 55011, Class A, Group 1 30 MHz to 230 MHz 230 MHz to 1 GHz	 <40dB (μV/m) quasi-peak; measured at 10m <47dB (μV/m) quasi-peak; measured at 10m

Environmental conditions

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, 2 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)

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

High Potential Isolation Test	
24 V/5 V nominal circuits 115/230 V circuits to ground 115/230 V circuits to 115/230 V circuits 115 V/230V circuits to 24 V/5 V circuits	520 VDC (type test of optical isolation boundaries) 1,500 VAC routine test/1950 VDC type test 1,500 VAC routine test/1950 VDC type test 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.

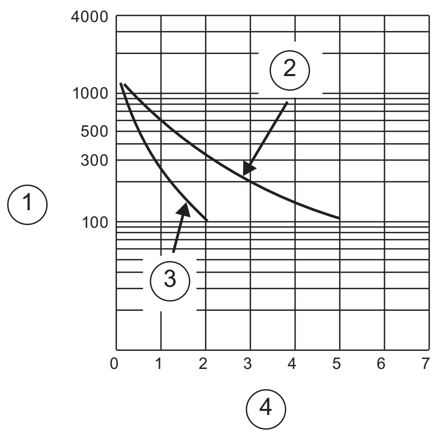
Rated voltages

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. You must plan for this, especially if you are using devices which respond to short duration pulses.

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 CPUs

A.2.1 CPU 1211C Specifications

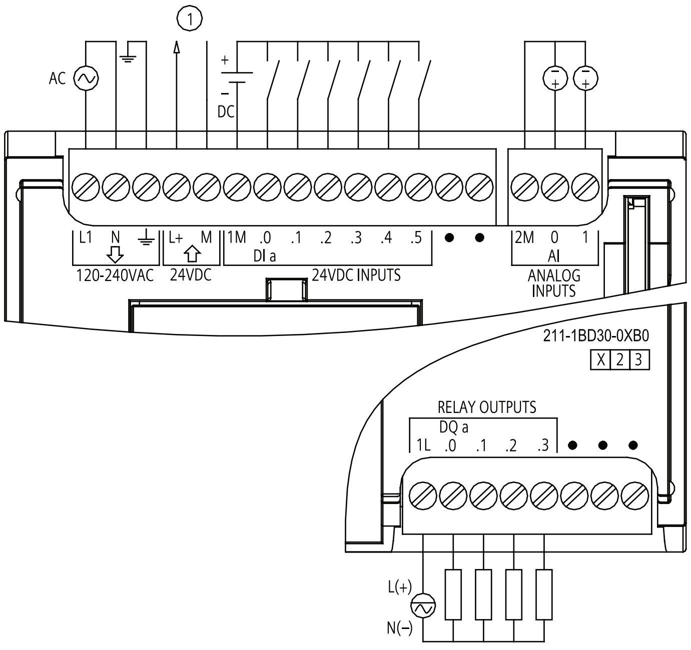
Technical Specifications			
Model	CPU 1211C AC/DC/Relay	CPU 1211C DC/DC/Relay	CPU 1211C DC/DC/DC
Order number (MLFB)	6ES7 211-1BD30-0XB0	6ES7 211-1HD30-0XB0	6ES7 211-1AD30-0XB0
General			
Dimensions W x H x D (mm)	90 x 100 x 75		
Weight	420 grams	380 grams	370 grams
Power dissipation	10 W	8 W	
Current available (CM bus)	750 mA max. (5 VDC)		
Current available (24 VDC)	300 mA max. (sensor power)		
Digital input current consumption (24VDC)	4 mA/input used		
CPU Features			
User memory	25 Kbytes Work memory / 1 Mbytes Load memory / 2 Kbytes Retentive memory		
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		
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		
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)		
Performance			
Boolean execution speed	0.1 μs/instruction		
Move Word execution speed	12 μs/instruction		
Real Math execution speed	18 μs/instruction		
Communication			
Number of ports	1		
Type	Ethernet		

Technical Specifications			
Model	CPU 1211C AC/DC/Relay	CPU 1211C DC/DC/Relay	CPU 1211C DC/DC/DC
Connections	• 3 for HMI • 1 for programming device • 8 for Ethernet instructions in the user program • 3 for CPU-to-CPU		
Data rates	10/100 Mb/s		
Isolation (external signal to PLC logic)	Transformer isolated, 1500 VDC		
Cable type	CAT5e shielded		
Power supply			
Voltage range	85 to 264 VAC	20.4 to 28.8 VDC	
Line frequency	47 to 63 Hz	--	
Input current CPU only at max. load CPU with all expansion accessories at max. load	 60 mA at 120 VAC 30 mA at 240 VAC 180 mA at 120 VAC 90 mA at 240 VAC	 300 mA at 24 VDC 900 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		
Sensor power			
Voltage range	20.4 to 28.8 VDC	L+ minus 4 VDC min.	
Output current rating (max.)	300 mA (short circuit protected)		
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	Same as input line	
Isolation (CPU logic to sensor power)	Not isolated		
Digital inputs			
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		

Technical Specifications			
Model	CPU 1211C AC/DC/Relay	CPU 1211C DC/DC/Relay	CPU 1211C DC/DC/DC
Cable length (meters)	500 shielded, 300 unshielded, 50 shielded for HSC inputs		
Analog inputs			
Number of inputs	2		
Type	Voltage (single-ended)		
Range	0 to 10 V		
Full-scale range (data word)	0 to 27648 (refer to Analog input representation for voltage (Page 306))		
Overshoot range (data word)	27,649 to 32,511 (refer to Analog input representation for voltage (Page 306))		
Overflow (data word)	32,512 to 32767 (refer to Analog input representation for voltage (Page 306))		
Resolution	10 bits		
Maximum withstand voltage	35 VDC		
Smoothing	None, Weak, Medium, or Strong (refer to Analog input response times (Page 306) for step response times)		
Noise rejection	10, 50, or 60 Hz (refer to Analog input response times (Page 306) for sample rates)		
Impedance	≥100 KΩ		
Isolation (field side to logic)	None		
Accuracy (25°C / 0 to 55°C)	3.0% / 3.5% of full-scale		
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, shielded twisted pair		
Digital outputs			
Number of outputs	4		
Type	Relay, dry contact		Solid state - MOSFET
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		
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
Switching delay (Qa.0 to Qa.3)	10 ms max.		1.0 μs max., off to on 3.0 μs max., on to off

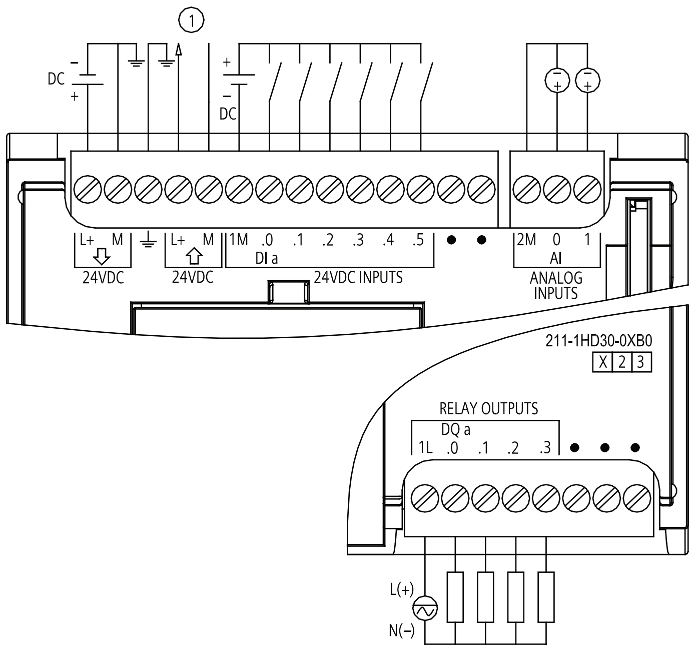
Technical Specifications			
Model	CPU 1211C AC/DC/Relay	CPU 1211C DC/DC/Relay	CPU 1211C DC/DC/DC
Pulse Train Output rate (Qa.0 and Qa.2)	Not recommended		100 KHz max., 2 Hz min.
Lifetime mechanical (no load)	10,000,000 open/close cycles		--
Lifetime contacts at rated load	100,000 open/close cycles		--
Behavior on RUN to STOP	Last value or substitute value (default value 0)		
Number of Outputs On simultaneously	4		
Cable length (meters)	500 shielded, 150 unshielded		

Wiring Diagrams



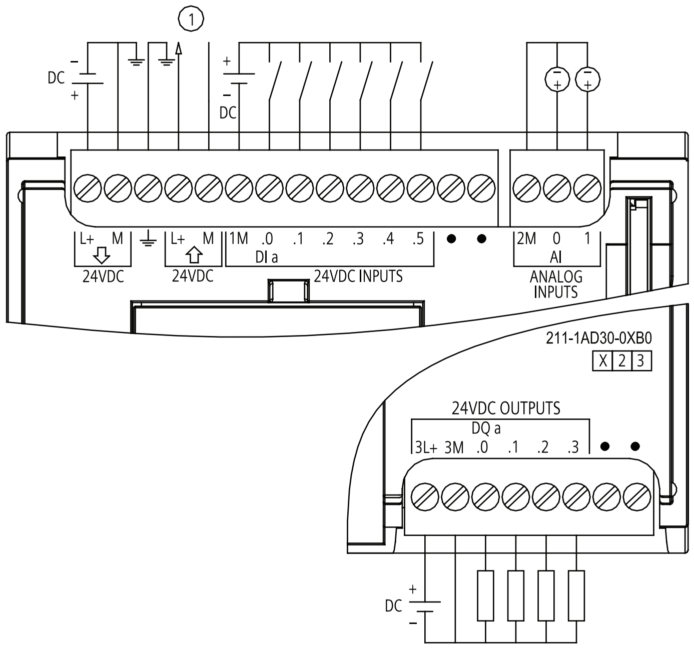
① 24 VDC Sensor Power Out

Figure A-1 CPU 1211C AC/DC/Relay (6ES7 211-1BD30-0XB0)



① 24 VDC Sensor Power Out

Figure A-2 CPU 1211C DC/DC/Relay (6ES7 211-1HD30-0XB0)



① 24 VDC Sensor Power Out

Figure A-3 CPU 1211C DC/DC/DC (6ES7 211-1AD30-0XB0)

A.2.2 CPU 1212C Specifications

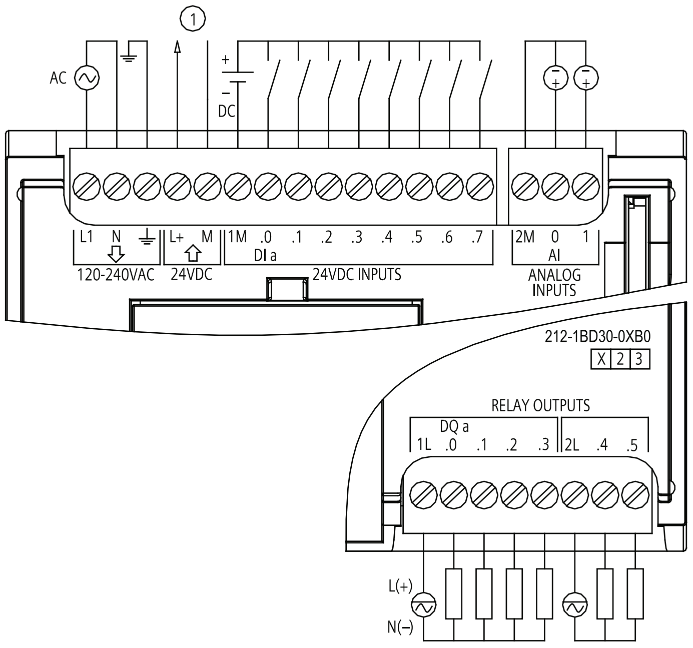
Technical Specifications			
Model	**CPU 1212C AC/DC/Relay**	**CPU 1212C DC/DC/Relay**	**CPU 1212C DC/DC/DC**
Order number (MLFB)	6ES7 212-1BD30-0XB0	6ES7 212-1HD30-0XB0	6ES7 212-1AD30-0XB0
General			
Dimensions W x H x D (mm)	90 x 100 x 75		
Weight	425 grams	385 grams	370 grams
Power dissipation	11 W	9 W	
Current available (SM and CM bus)	1000 mA max. (5 VDC)		
Current available (24 VDC)	300 mA max. (sensor power)		
Digital input current consumption (24 VDC)	4 mA/input used		
CPU Features			
User memory	25 Kbytes Work memory / 1 Mbytes Load memory/ 2 Kbytes Retentive memory		
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		
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		
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)		
Performance			
Boolean execution speed	0.1 μs/instruction		
Move Word execution speed	12 μs/instruction		
Real Math execution speed	18 μs/instruction		
Communication			
Number of ports	1		
Type	Ethernet		
Connections	• 3 for HMI • 1 for programming device • 8 for Ethernet instructions in the user program • 3 for CPU-to-CPU		

Technical Specifications			
Model	**CPU 1212C AC/DC/Relay**	**CPU 1212C DC/DC/Relay**	**CPU 1212C DC/DC/DC**
Data rates	10/100 Mb/s		
Isolation (external signal to PLC logic)	Transformer isolated, 1500 VDC		
Cable type	CAT5e shielded		
Power supply			
Voltage range	85 to 264 VAC	20.4 to 28.8 VDC	
Line frequency	47 to 63 Hz	--	
Input current CPU only at max. load CPU with all expansion accessories at max. load	80 mA at 120 VAC 40 mA at 240 VAC 240 mA at 120 VAC 120 mA at 240 VAC	400 mA at 24 VDC 1200 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		
Sensor power			
Voltage range	20.4 to 28.8 VDC	L+ minus 4 VDC min.	
Output current rating (max.)	300 mA (short circuit protected)		
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	Same as input line	
Isolation (CPU logic to sensor power)	Not isolated		
Digital inputs			
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 shielded, 300 unshielded, 50 shielded for HSC inputs		
Analog inputs			
Number of inputs	2		
Type	Voltage (single-ended)		
Range	0 to 10 V		

Technical Specifications			
Model	CPU 1212C AC/DC/Relay	CPU 1212C DC/DC/Relay	CPU 1212C DC/DC/DC
Full-scale range (data word)	0 to 27648 (Refer to Analog input representation for voltage (Page 306))		
Overshoot range (data word)	27,649 to 32,511 (Refer to Analog input representation for voltage (Page 306))		
Overflow (data word)	32,512 to 32767 (Refer to Analog input representation for voltage (Page 306))		
Resolution	10 bits		
Maximum withstand voltage	35 VDC		
Smoothing	None, Weak, Medium, or Strong (refer to Analog input response times (Page 306) for step response times)		
Noise rejection	10, 50, or 60 Hz (refer to Analog input response times (Page 306) for sample rates)		
Impedance	≥100 KΩ		
Isolation (field side to logic)	None		
Accuracy (25°C / 0 to 55°C)	3.0% / 3.5% of full-scale		
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 twisted and shielded		
Digital outputs			
Number of outputs	6		
Type	Relay, dry contact		Solid state - MOSFET
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		
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
Pulse Train Output rate (Qa.0 and Qa.2)	Not recommended		100 KHz max., 2 Hz min.
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)		

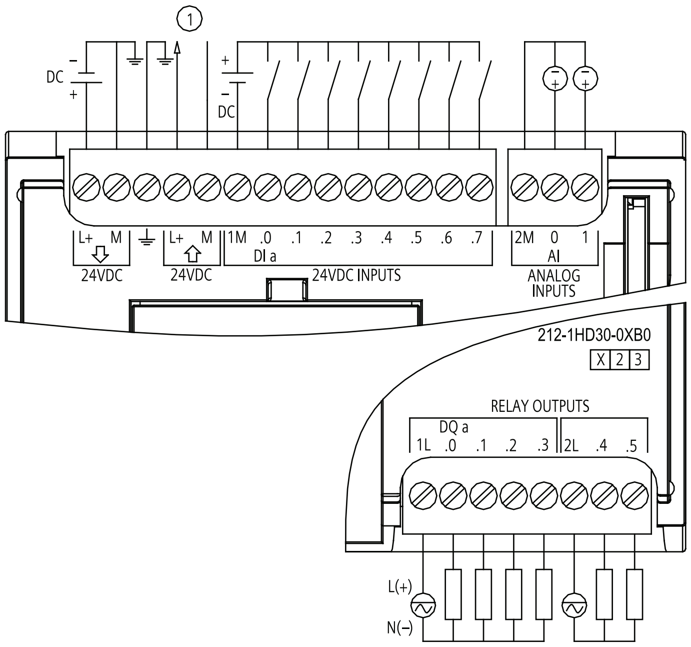
Technical Specifications			
Model	CPU 1212C AC/DC/Relay	CPU 1212C DC/DC/Relay	CPU 1212C DC/DC/DC
Number of Outputs On simultaneously	6		
Cable length (meters)	500 shielded, 150 unshielded		

Wiring Diagrams



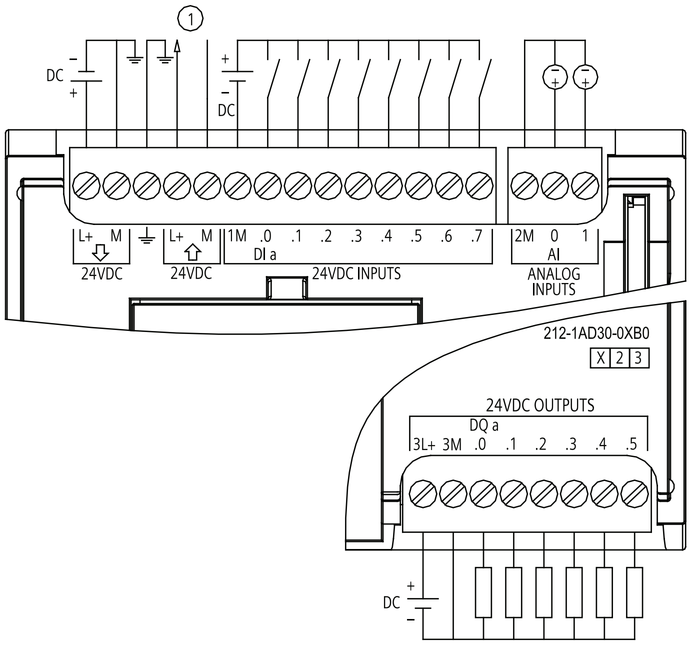
① 24 VDC Sensor Power Out

Figure A-4 CPU 1212C AC/DC Relay (6ES7 212-1BD30-0XB0)



① 24 VDC Sensor Power Out

Figure A-5 CPU 1212C DC/DC/Relay (6ES7 212-1HD30-0XB0)



① 24 VDC Sensor Power Out

Figure A-6 CPU 1212C DC/DC/DC (6ES7 212-1AD30-0XB0)

A.2.3 CPU 1214C Specifications

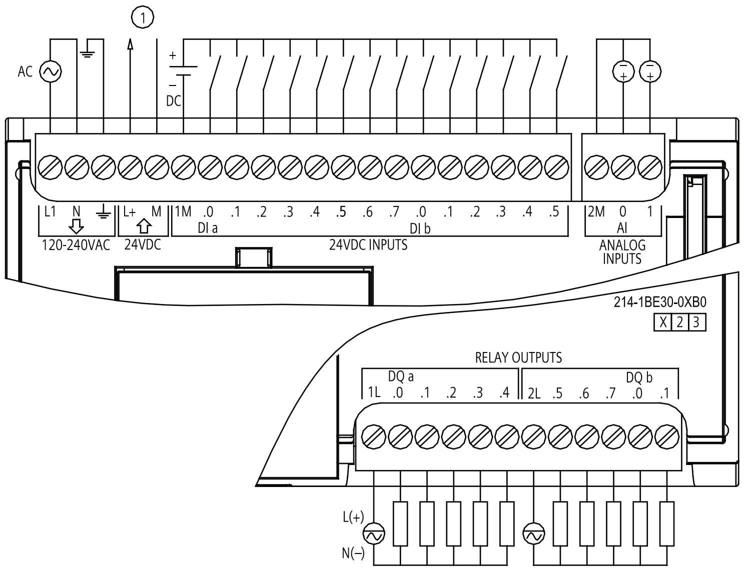
Technical Specifications			
Model	**CPU 1214C AC/DC/Relay**	**CPU 1214C DC/DC/Relay**	**CPU 1214C DC/DC/DC**
Order number (MLFB)	6ES7 214-1BE30-0XB0	6ES7 214-1HE30-0XB0	6ES7 214-1AE30-0XB0
General			
Dimensions W x H x D (mm)	110 x 100 x 75		
Weight	475 grams	435 grams	415 grams
Power dissipation	14 W	12 W	
Current available (SM and CM bus)	1600 mA max. (5 VDC)		
Current available (24 VDC)	400 mA max. (sensor power)		
Digital input current consumption (24VDC)	4 mA/input used		
CPU Features			
User memory	50 Kbytes Work memory / 2 Mbytes Load memory/ 2 Kbytes Retentive memory		
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		
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		
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)		
Performance			
Boolean execution speed	0.1 μs/instruction		
Move Word execution speed	12 μs/instruction		
Real Math execution speed	18 μs/instruction		
Communication			
Number of ports	1		
Type	Ethernet		
Connections	• 3 for HMI • 1 for programming device • 8 for Ethernet instructions in the user program • 3 for CPU-to-CPU		

Technical Specifications			
Model	**CPU 1214C AC/DC/Relay**	**CPU 1214C DC/DC/Relay**	**CPU 1214C DC/DC/DC**
Data rates	10/100 Mb/s		
Isolation (external signal to PLC logic)	Transformer isolated, 1500 VDC		
Cable type	CAT5e shielded		
Power supply			
Voltage range	85 to 264 VAC	20.4 to 28.8 VDC	
Line frequency	47 to 63 Hz	--	
Input current CPU only at max. load CPU with all expansion accessories at max. load	 100 mA at 120 VAC 50 mA at 240 VAC 300 mA at 120 VAC 150 mA at 240 VAC	 500 mA at 24 VDC 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		
Sensor power			
Voltage range	20.4 to 28.8 VDC	L+ minus 4 VDC min.	
Output current rating (max.)	400 mA (short circuit protected)		
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	Same as input line	
Isolation (CPU logic to sensor power)	Not isolated		
Digital inputs			
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 shielded, 300 unshielded, 50 shielded for HSC inputs		
Analog inputs			
Number of inputs	2		
Type	Voltage (single-ended)		

Technical Specifications			
Model	**CPU 1214C AC/DC/Relay**	**CPU 1214C DC/DC/Relay**	**CPU 1214C DC/DC/DC**
Range	0 to 10 V		
Full-scale range (data word)	0 to 27648 (Refer to Analog input representation for voltage (Page 306))		
Overshoot range (data word)	27,649 to 32,511 (Refer to Analog input representation for voltage (Page 306))		
Overflow (data word)	32,512 to 32767 (Refer to Analog input representation for voltage (Page 306))		
Resolution	10 bits		
Maximum withstand voltage	35 VDC		
Smoothing	None, Weak, Medium, or Strong (refer to Analog input response time (Page 306) for step response times)		
Noise rejection	10, 50, or 60 Hz (refer to Analog input response time (Page 306) for sample rates)		
Impedance	≥100 KΩ		
Isolation (field side to logic)	None		
Accuracy (25°C / 0 to 55°C)	3.0% / 3.5% of full-scale		
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 twisted and shielded		
Digital outputs			
Number of outputs	10		
Type	Relay, dry contact		Solid state - MOSFET
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		
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
Pulse Train Output rate (Qa.0 and Qa.2)	Not recommended		100 KHz max., 2 Hz min.
Lifetime mechanical (no load)	10,000,000 open/close cycles		--
Lifetime contacts at rated load	100,000 open/close cycles		--

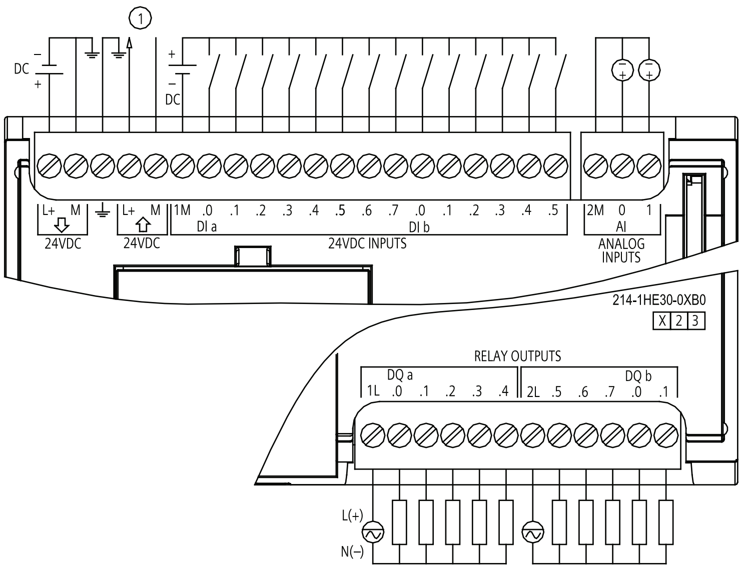
Technical Specifications			
Model	**CPU 1214C AC/DC/Relay**	**CPU 1214C DC/DC/Relay**	**CPU 1214C DC/DC/DC**
Behavior on RUN to STOP	Last value or substitute value (default value 0)		
Number of Outputs On simultaneously	10		
Cable length (meters)	500 shielded, 150 unshielded		

Wiring Diagrams



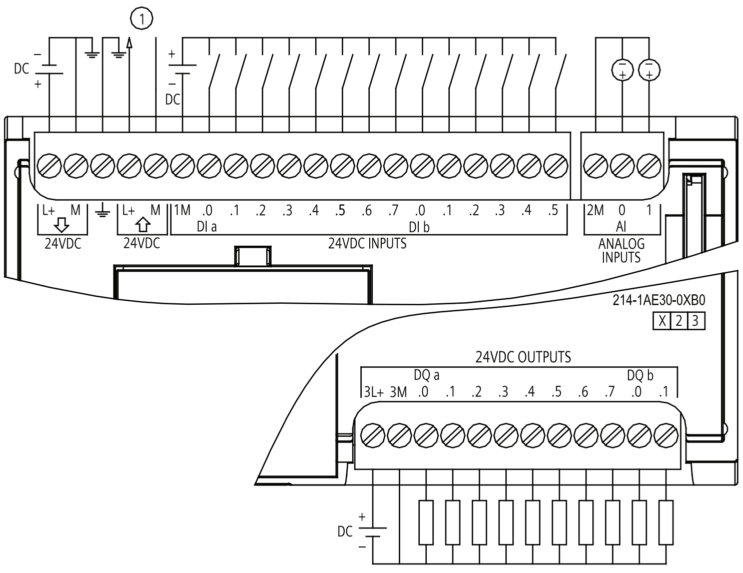
① 24 VDC Sensor Power Out

Figure A-7 CPU 1214C AC/DC/Relay (6ES7 214-1BE30-0XB0)



① 24 VDC Sensor Power Out

Figure A-8 CPU 1214C DC/DC/Relay (6ES7 214-1HE30-0XB0)



① 24 VDC Sensor Power Out

Figure A-9 CPU 1214C DC/DC/DC (6ES7 214-1AE30-0XB0)

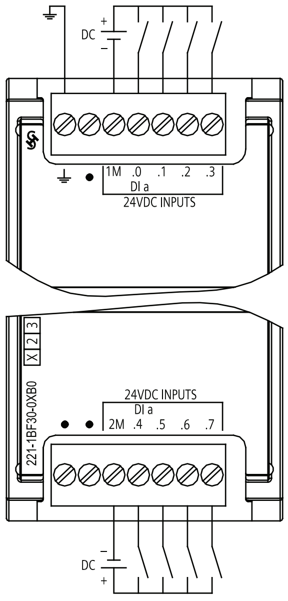
A.3 Digital signal modules (SMs)

A.3.1 SM 1221 Digital Input Specifications

Technical Specifications		
Model	**SM 1221 DI 8x24VDC**	**SM 1221 DI 16x24VDC**
Order number (MLFB)	6ES7 221-1BF30-0XB0	6ES7 221-1BH30-0XB0
General		
Dimensions W x H x D (mm)	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
Digital inputs		
Number of inputs	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	4
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
Cable length (meters)	500 shielded, 300 unshielded	

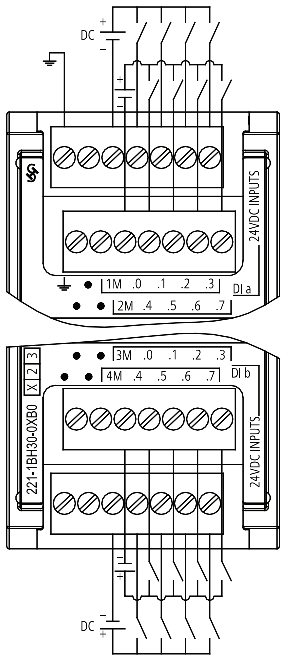
Wiring diagrams

SM 1221 DI 8 x 24 VDC



6ES7 221-1BF30-0XB0

SM 1221 DI 16 x 24 VDC



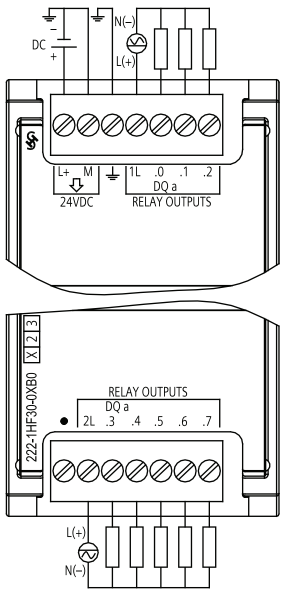
6ES7 221-1BH30-0XB0

A.3.2 SM 1222 Digital Output Specifications

Technical Specifications				
Model	SM 1222 DQ 8xRelay	SM1222 DQ 16xRelay	SM1222 DQ 8x24VDC	SM1222 DQ 16x24VDC
Order number (MLFB)	6ES7 222-1HF30-0XB0	6ES7 222-1HH30-0XB0	6ES7 222-1BF30-0XB0	6ES7 222-1BH30-0XB0
General				
Dimensions W x H x D (mm)	45 x 100 x 75			
Weight	190 grams	260 grams	180 grams	220 grams
Power dissipation	4.5 W	8.5 W	1.5 W	2.5 W
Current consumption (SM Bus)	120 mA	135 mA	120 mA	140 mA
Current consumption (24 VDC)	11 mA / Relay coil used		--	
Digital Outputs				
Number of outputs	8	16	8	16
Type	Relay, dry contact		Solid state - MOSFET	
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			
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 (max.)	10 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	
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 shielded, 150 unshielded			

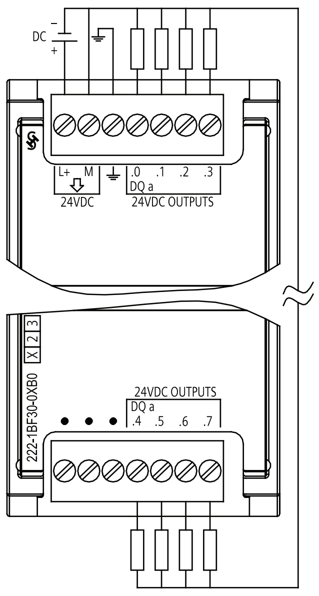
Wiring Diagrams

SM 1222 DQ 8 x Relay



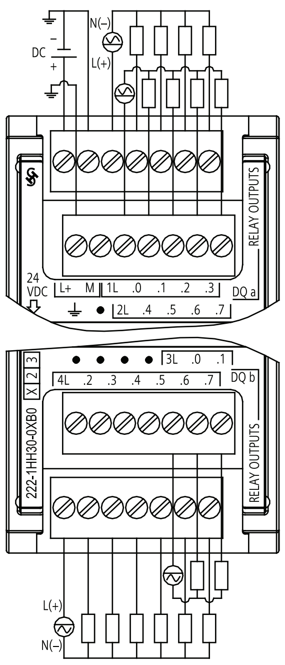
6ES7 222-1HF30-0XB0

SM 1222 DQ 8 x 24 VDC



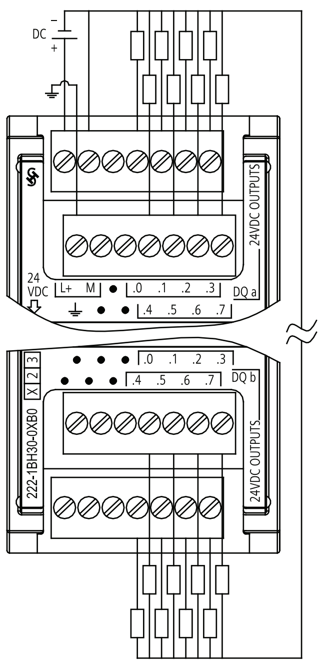
6ES7 222-1BF30-0XB0

SM 1222 DQ 16 x Relay



6ES7 222-1HH30-0XB0

SM 1222 DQ 16 x 24 VDC



6ES7 222-1BH30-0XB0

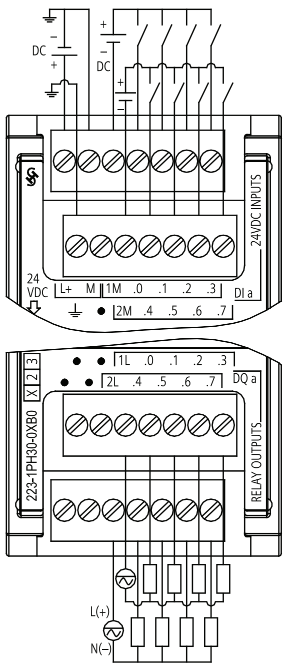
A.3.3 SM 1223 Digital Input/Output Specifications

Technical Specifications				
Model	SM 1223 DI 8x24 VDC, DQ 8xRelay	SM 1223 DI 16x24 VDC, DQ 16xRelay	SM 1223 DI 8x24 VDC, DQ 8x24 VDC	SM 1223 DI 16x24 VDC, DQ16x24 VDC
Order number (MLFB)	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	
Digital Inputs				
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 shielded, 300 unshielded			
Digital Outputs				
Number of outputs	8	16	8	16
Type	Relay, dry contact		Solid state - MOSFET	
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 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		--	

Technical Specifications				
Model	SM 1223 DI 8x24 VDC, DQ 8xRelay	SM 1223 DI 16x24 VDC, DQ 16xRelay	SM 1223 DI 8x24 VDC, DQ 8x24 VDC	SM 1223 DI 16x24 VDC, DQ16x24 VDC
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	
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 shielded, 150 unshielded			

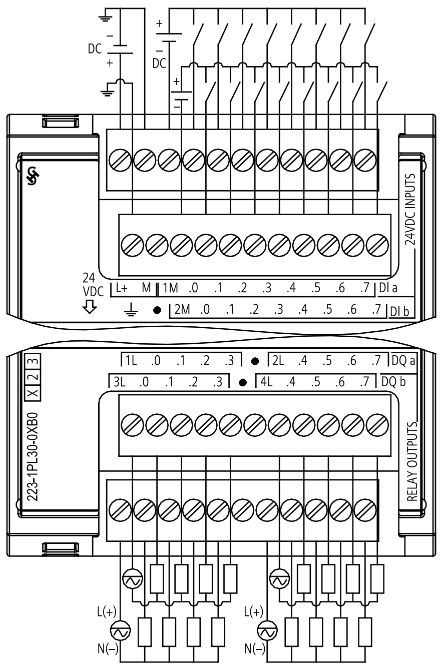
Wiring diagrams

SM 1223 DI 8 x 24 VDC, DQ 8 x Relay



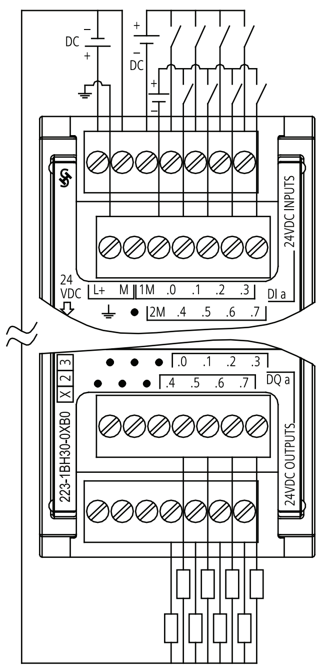
6ES7 223-1PH30-0XB0

SM1223 DI 16 x 24 VDC, DQ 16 x Relay


DC
DC
24VDC INPUTS
24 VDC
L+ M 1M .0 .1 .2 .3 .4 .5 .6 .7 DI a
2M .0 .1 .2 .3 .4 .5 .6 .7 DI b
X 2 3
1L .0 .1 .2 .3 2L .4 .5 .6 .7 DQ a
3L .0 .1 .2 .3 4L .4 .5 .6 .7 DQ b
223-1PL30-0XB0
RELAY OUTPUTS
L(+)
N(-)
L(+)
N(-)

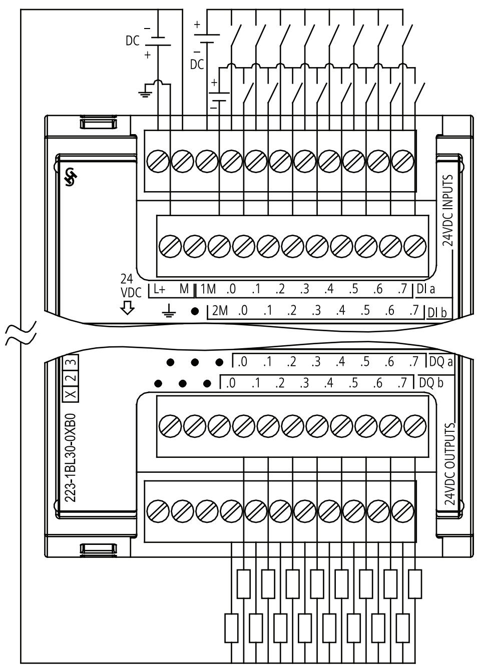
6ES7 223-1PL30-0XB0

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


DC
DC
24 VDC
L+ M 1M .0 .1 .2 .3 .4 .5 .6 .7 DI a
2M .0 .1 .2 .3 .4 .5 .6 .7 DI b
24VDC INPUTS
X 2 3
.0 .1 .2 .3 .4 .5 .6 .7 DQ a
.0 .1 .2 .3 .4 .5 .6 .7 DQ b
223-1BL30-0XB0
24VDC OUTPUTS

6ES7 223-1BL30-0XB0

A.4 Analog signal modules (SMs)

A.4.1 SM 1231, SM 1232, SM 1234 Analog Specifications

Technical Specifications			
Model	**SM 1231 AI 4x13bit**	**SM 1231 AI 8x13bit**	**SM 1234 AI 4x13bit AQ 2x14bit**
Order number (MLFB)	6ES7 231-4HD30-0XB0	6ES7 231-4HF30-0XB0	6ES7 234-4HE30-0XB0
General			
Dimensions W x H x D (mm)	45 x 100 x 75	45 x 100 x 75	45 x 100 x 75
Weight	180 grams	180 grams	220 grams
Power dissipation	1.5 W	1.5 W	2.0 W
Current consumption (SM Bus)	80 mA	90 mA	80 mA
Current consumption (24 VDC)	45 mA	45 mA	60 mA (no load)
Analog Inputs			
Number of inputs	4	8	4
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 Analog input representation for voltage, Analog input representation for current (Page 306))		
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 Analog input representation for voltage, Analog input representation for current (Page 306))		
Resolution	12 bits + sign bit		
Maximum withstand voltage/current	±35 V / ±40 mA		
Smoothing	None, weak, medium, or strong (refer to Analog input response times (Page 306) for step response times)		
Noise rejection	400, 60, 50, or 10 Hz (refer to Analog input response times (Page 306) for sample rates)		
Impedance	≥ 9 MΩ (voltage) / 250 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 meters, twisted and shielded		

Technical Specifications			
Model	**SM 1231 AI 4x13bit**	**SM 1231 AI 8x13bit**	**SM 1234 AI 4x13bit AQ 2x14bit**
Diagnostics			
Overflow/underflow	Yes[1]	Yes[1]	Yes[1]
Short to ground (voltage mode only)	Not applicable	Not applicable	Yes on outputs
Wire break (current mode only)	Not applicable	Not applicable	Yes on outputs
24 VDC low voltage	Yes	Yes	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.

Technical Specifications			
Model	**SM 1232 AQ 2x14bit**	**SM 1232 AQ 4x14bit**	**SM 1234 AI 4x13bit AQ 2x14bit**
Order number (MLFB)	6ES7 232-4HB30-0XB0	6ES7 232-4HD30-0XB0	6ES7 234-4HE30-0XB0
General			
Dimensions W x H x D (mm)	45 x 100 x 75	45 x 100 x 75	45 x 100 x 75
Weight	180 grams	180 grams	220 grams
Power dissipation	1.5 W	1.5 W	2.0 W
Current consumption (SM Bus)	80 mA	80 mA	80 mA
Current consumption (24 VDC)	45 mA (no load)	45 mA (no load)	60 mA (no load)
Analog Outputs			
Number of outputs	2	4	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 Analog output representation for voltage and Analog output representation for current) (Page 306)		
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 meters twisted and shielded		
Diagnostics			
Overflow/underflow	Yes	Yes	Yes[1]
Short to ground (voltage mode only)	Yes	Yes	Yes on outputs
Wire break (current mode only)	Yes	Yes	Yes on outputs
24 VDC low voltage	Yes	Yes	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.

Analog input response time

SM Analog Modules Step Response (ms)				
0V to 10V measured at 95%				
Smoothing Selection	Rejection Frequency			
	400 Hz	60 Hz	50 Hz	10 Hz
None	4	18	22	100
Weak	9	52	63	320
Medium	32	203	241	1200
Strong	61	400	483	2410
Sample Rate • 4 channels • 8 channels	 • 0.625 • 1.25	 • 4.17 • 4.17	 • 5 • 5	 • 25 • 25

CPU Analog Input Step Response (ms)			
0V to 10V measured at 95%			
Smoothing Selection	Rejection Frequency		
	60 Hz	50 Hz	10 Hz
None	63	65	130
Weak	84	93	340
Medium	221	258	1210
Strong	424	499	2410
Sample Rate	**4.17**	**5**	**25**

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	11.851V	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			

Analog input representation for current

System	Current Measuring Range		
Decimal	**Hexadecimal**	**0 mA to 20 mA**	
32767	7FFF	23.70 mA	Overflow
32512	7F00		
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	FFFF		Undershoot range
-4864	ED00	-3.52 mA	
-4865	ECFF		Underflow
-32768	8000		

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.

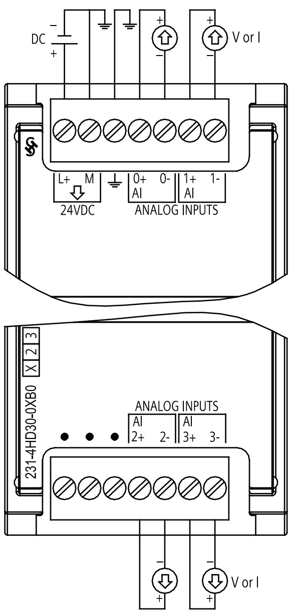
Analog output representation for current

System		Current Output Range	
Decimal	Hexadecimal	± 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	FFFF		Undershoot range
-32512	8100		
-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.

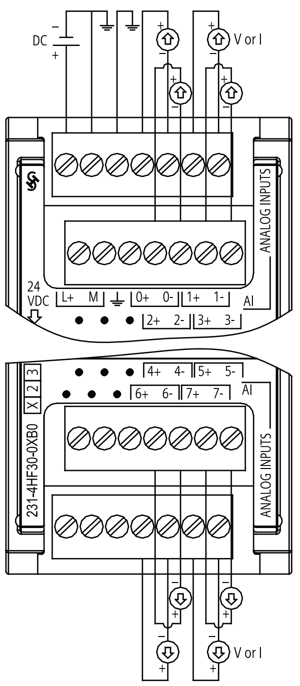
Wiring Diagrams

SM 1231 AI 4 x 13 bit



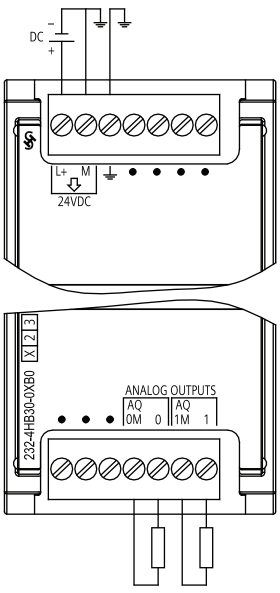
6ES7 231-4HD30-0XB0

SM 1231 AI 8 x 13 bit



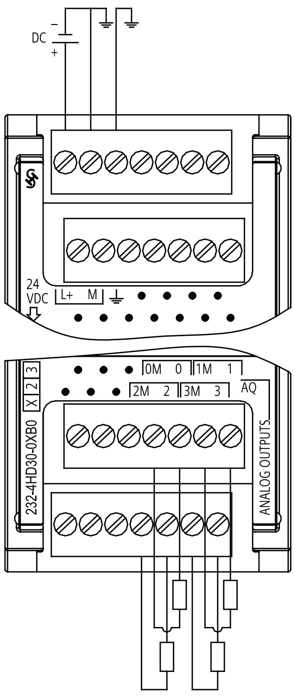
6ES7 231-4HF30-0XB0

SM 1232 AQ 2 x 14 bit



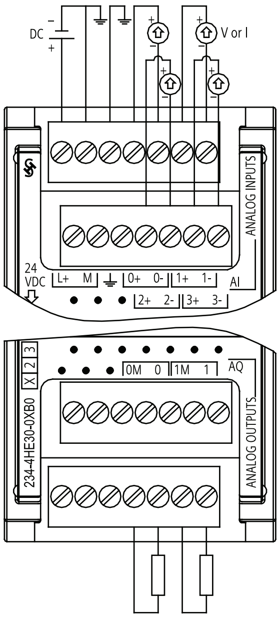
6ES7 232-4HB30-0XB0

SM 1232 AQ 4 x 14 bit



6ES7 232-4HD30-0XB0

SM 1234 AI 4 x 13 Bit / AQ 2 x 14 bit



6ES7 234-4HE30-0XB0

A.5 Signal boards (SBs)

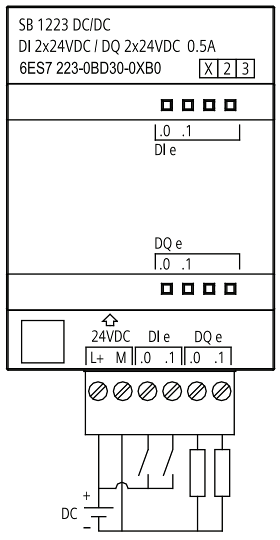
A.5.1 SB 1223 2 X 24 VDC Input / 2 X 24 VDC Output Specifications

Digital signal board specifications

Technical Data	
Model	**SB 1223 DI 2x24VDC, DQ 2x24VDC**
Order number (MLFB)	6ES7 223-0BD30-0XB0
General	
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
Digital inputs	
Number of inputs	2
Type	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
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
Digital Outputs	
Number of outputs	2
Output type	Solid state - MOSFET
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 rate	20 KHz max., 2 Hz min.

Technical Data	
Model	**SB 1223 DI 2x24VDC, DQ 2x24VDC**
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 shielded, 150 unshielded

SB 1223 2 x 24 VDC Input / 2 x 24 VDC Output wiring diagram

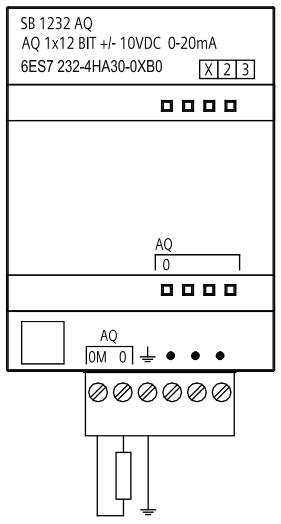


A.5.2 SB 1232 1 Analog Output Specifications

Analog signal board specifications

Technical Data	
Model	**SB 1223 AQ 1x12bit**
Order no. (MLFB)	6ES7 232-4HA30-0XB0
General	
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)
Analog Outputs	
Number of outputs	1
Type	Voltage or current
Range	±10 V or 0 to 20 mA
Resolution	Voltage: 12 bits Current: 11 bits
Full scale range (data word)	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 meters, twisted and shielded
Diagnostics	
Overflow/underflow	Yes
Short to ground (voltage mode only)	Yes
Wire break (current mode only)	Yes

SB 1232 1 x Analog Output wiring diagram

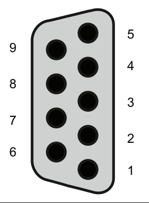


A.6 Communication modules (CMs)

A.6.1 CM 1241 RS485 Specifications

Table A- 1 Communication Module CM 1241 RS485

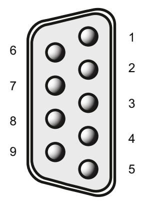
Technical Data	
Order no. (MLFB)	6ES7 241-1CH30-0XB0
Dimensions and weight	
Dimensions	30 x 100 x 75 mm
Weight	150 grams
Transmitter and Receiver	
Common mode voltage range	-7 V to +12 V, 1 second, 3 VRMS continuous
Transmitter differential output voltage	2 V min. at R_L = 100 Ω 1.5 V min. at R_L = 54 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.
Power supply specification	
Power loss (dissipation)	1.1 W
From +5 VDC	220 mA

Pin	Description	Connector (female)	Pin	Description
1 GND	Logic or communication ground		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	Request to send (TTL level): Output		9	Not connected
5 GND	Logic or communication ground		SHELL	Chassis ground

A.6.2 CM 1241 RS232 Specifications

Communication Module CM 1241 RS232

Technical Data	
Order no. (MLFB)	6ES7 241-1AH30-0XB0
Dimensions and weight	
Dimensions	30 x 100 x 75 mm
Weight	150 grams
Transmitter and Receiver	
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.
Power supply specification	
Power loss (dissipation)	1.1 W
From +5 VDC	220 mA

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.7 SIMATIC memory cards

Memory card specifications

Order Number	Capacity
6ES7 954-8LF00-0AA0	24 MB
6ES7 954-8LB00-0AA0	2 MB

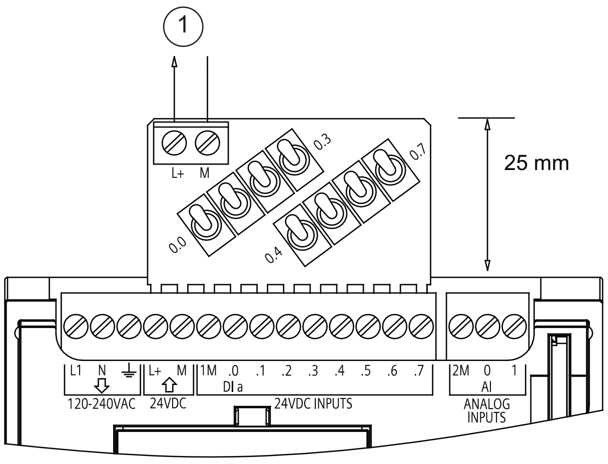
A.8 Input simulators

Model	8 Position Simulator	14 Position Simulator
Order number (MLFB)	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.

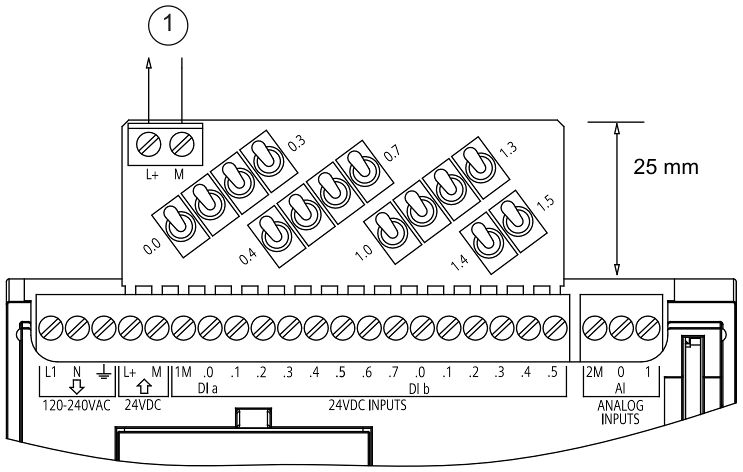
8 Position Simulator



① 24 VDC sensor power out

6ES7 274-1XF30-0XA0

14 Position Simulator



① 24 VDC sensor power out

6ES7 274-1XH30-0XA0

A.9 I/O expansion cable

Technical Data	
Order no (MLFB)	6ES7 290-6AA30-0XA0
Cable length	2 m
Weight	200 g

The I/O expansion cable has a male and female connector.

1. Connect the male connector to the bus connector on the right side of the signal module.
2. Connect the female connector to the bus connector on the left side of the signal module.
   – Slip the hook extension of the female connector into the housing at the bus connector
   – Push the female connector into the bus connector.

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 three 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 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 279).

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.

B.2 Calculating a sample power requirement

The following example shows a sample calculation of the power requirements for a PLC that includes a CPU 1214C AC/DC/Relay, 3 x SM 1223 8 DC In/8 Relay Out, and 1 x SM 1221 8 DC In. This example has a total of 46 inputs and 34 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 448 mA and the CPU provides only 400 mA. This installation requires an additional source of at least 48 mA at 24 VDC power to operate all the included 24 VDC inputs and outputs.

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
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	-	8 * 4 mA = 32 mA
Total requirements	540 mA	448 mA
Equals		
Current balance	**5 VDC**	**24 VDC**
Current balance total	1060 mA	(48 mA)

B.3 Calculating your power requirement

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 279) for the power budgets of your CPU model and the power requirements of your signal modules.

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		

C Order numbers

CPUs		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

Signal modules, communication modules, and signal boards		Order Number
Signal modules	SM 1221 8 x 24 VDC Input	6ES7 221-1BF30-0XB0
	SM 1221 16 x 24 VDC Input	6ES7 221-1BH30-0XB0
	SM 1222 8 x 24 VDC Output	6ES7 222-1BF30-0XB0
	SM 1222 16 x 24 VDC Output	6ES7 222-1BH30-0XB0
	SM 1222 8 x Relay Output	6ES7 222-1HF30-0XB0
	SM 1222 16 x Relay Output	6ES7 222-1HH30-0XB0
	SM 1223 8 x 24 VDC Input / 8 x 24 VDC Output	6ES7 223-1BH30-0XB0
	SM 1223 16 x 24 VDC Input / 16 x 24 VDC Output	6ES7 223-1BL30-0XB0
	SM 1223 8 x 24 VDC Input / 8 x Relay Output	6ES7 223-1PH30-0XB0
	SM 1223 16 x 24 VDC Input / 16 x Relay Output	6ES7 223-1PL30-0XB0
	SM 1231 4 x Analog Input	6ES7 231-4HD30-0XB0
	SM 1231 8 x Analog Input	6ES7 231-4HF30-0XB0
	SM 1232 2 x Analog Output	6ES7 232-4HB30-0XB0
	SM 1232 4 x Analog Output	6ES7 232-4HD30-0XB0
	SM 1234 4 x Analog Input / 2 x Analog Output	6ES7 234-4HE30-0XB0
Communication modules	CM 1241 RS232	6ES7 241-1AH30-0XB0
	CM 1241 RS485	6ES7 241-1CH30-0XB0
Signal boards	SB 1223 2 x 24 VDC Input / 2 x 24 VDC Output	6ES7 223-0BD30-0XB0
	SB 1232 1 Analog Output	6ES7 232-4HA30-0XB0

HMI devices	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

Programming package	Order Number
STEP 7 Basic v10.5	6ES7 822-0AA0-0YA0

Memory cards, other hardware, and spare parts		Order Number
Memory Cards	SIMATIC MC 2 MB	6ES7 954-8LB00-0AA0
	SIMATIC MC 24 MB	6ES7 954-8LF00-0AA0
Other hardware	PSU 1200 power supply	6EP1 332-1SH71
	CSM 1277 Ethernet switch - 4 ports	6GK7 277-1AA00-0AA0
	Simulator (1214C/1211C - 8 position)	6ES7 274-1XF30-0XA0
	Simulator (1214C - 14 position)	6ES7 274-1XH30-0XA0
	I/O Expansion cable, 2 m	6ES7 290-6AA30-0XA0
Spare Parts	Connector block, 7 terminal, Tin	6ES7 292-1AG30-0XA0
	Connector block, 8 terminal, Tin (4/pk)	6ES7 292-1AH30-0XA0
	Connector block, 11 terminal, Tin (4/pk)	6ES7 292-1AL30-0XA0
	Connector block, 12 terminal, Tin (4/pk)	6ES7 292-1AM30-0XA0
	Connector block, 14 terminal, Tin (4/pk)	6ES7 292-1AP30-0XA0
	Connector block, 20 terminal, Tin (4/pk)	6ES7 292-1AV30-0XA0
	Connector block, 3 terminal, Gold (4/pk)	6ES7 292-1BC0-0XA0
	Connector block, 6 terminal, Gold (4/pk)	6ES7 292-1BF30-0XA0
	Connector block, 7 terminal, Gold (4/pk)	6ES7 292-1BG30-0XA0
	Connector block, 11 terminal, Gold (4/pk)	6ES7 292-1BL30-0XA0

Documentation	Order Number
S7-1200 Programmable Controller System Manual • German • English • French • Spanish • Italian • Chinese	 • 6ES7 298-8FA30-8AH0 • 6ES7 298-8FA30-8BH0 • 6ES7 298-8FA30-8CH0 • 6ES7 298-8FA30-8DH0 • 6ES7 298-8FA30-8EH0 • 6ES7 298-8FA30-8KH0
S7-1200 Easy Book • German • English • French • Spanish • Italian • Chinese	 • 6ES7 298-8FA30-8AQ0 • 6ES7 298-8FA30-8BQ0 • 6ES7 298-8FA30-8CQ0 • 6ES7 298-8FA30-8DQ0 • 6ES7 298-8FA30-8EQ0 • 6ES7 298-8FA30-8KQ0

Index

A

Absolute value (ABS) instruction, 116
AC
    Inductive loads, 34
Accessing the online help, 16
Add (ADD) instruction, 113
Adding a device
    Unspecific CPU, 71
Analog I/O status indicators, 270
Analog signal module specifications, 306
Analog signal module voltage, 308
AND instruction, 127
ATEX approval, 280
ATTACH interrupt instruction, 167

B

Baud rate, 239
Bit logic, 93
Block
    Types, 37
Block call
    Basics, 37
    Calling as single instance or multi-instance, 85
Block move (MOVE_BLK) instruction, 119
Blocks
    data blocks (DBs), 37
    function blocks (FBs), 37
    functions (FCs), 37
    organization blocks (OBs), 37, 42
Break, 241, 243
Bus connector, 13

C

Calendar instructions, 132
CAN_DINT time delay interrupt instruction, 170
CE approval, 279
Ceiling (CEIL) instruction, 124
Character position
    message length, 245
Character sequence
    message end, 244
    message start, 243
Clearance for installation, 22
Clock
    time-of-day clock, 50
Clock instructions, 134
    read local time (RD_LOC_T), 134
    read system time (RD_SYS_T), 134
    write system time (WR_SYS_T), 134
CM 1241 RS232 specifications, 318
CM 1241 RS485 specifications, 317
Code block
    DB (data block), 86
    FB (function block), 84
    FC (function), 84
    Know-how protection, 89
Code blocks, 81
Communication
    flow control, 239
    hardware connection, 211
    IP address, 76, 217
    libraries, 237
    load, 48
    network, 210
    polling architecture, 247
    send and receive parameters, 241
Communication interfaces
    configuration, 238
    programming, 246
Communication module
    Add modules, 73
    Add new device, 70
    Device configuration, 69
Communication module (CM)
    Comparison chart, 12
    Installation, 29
    Overview, 14
    Removal, 29
Communication modules
    RS232 and RS485, 237
Communications module (CM), 248
    data reception, 262
    power requirements, 321
    specifications, 317
Communications module (CM), USS library, 177
Compare instructions, 111
Comparison chart
    CPU models, 12
    HMI devices, 19
Comparison chart of modules, 12
Configuration

communication interfaces, 238
Cycle time, 48
Discover, 71
HMI to CPU, 222
HSC (high-speed counter), 110
Industrial Ethernet port, 76, 217
IP address, 76, 217
PLC to PLC communication, 223
ports, 238
PROFINET, 76, 217
receive message, 242
Startup parameters, 38, 63
Configuring parameters
CPU, 72
Ethernet port, 76, 217
modules, 74
PROFINET, 76, 217
Connector
Installation and Removal, 31
Contact information, 3
Context-sensitive help, 16
Convert instructions, 123
Copy protection, 89
Counter
high-speed (HSC), 107
high-speed (HSC): configuring, 110
Counter instructions, 102
CPU
1211C specifications, 284
1211C wiring diagrams, 287
1212C specifications, 289
1212C wiring diagrams, 292
1214C specifications, 294
1214C wiring diagrams, 297
Add modules, 73
Add new device, 70
Comparison chart, 12
configuring communication to HMI, 221
configuring multiple, 223
Configuring parameters, 72
Creating a Program card, 66
Creating a transfer card, 63
Cycle time, 48
Device configuration, 69
download to device, 220
Empty transfer card, 53
Ethernet port, 76, 217
going online, 270
Grounding, 33
Inductive loads, 34
Installation procedures, 26
IP address, 76, 217
Isolation guidelines, 33
Lamp loads, 35
Lost password, 53
MAC address, 233
Network connection, 75
online, 271
Operating modes, 39
operating panel for online, 272
Overview, 11
Password protection, 52
Power budget, 22
power requirements, 321
PROFINET, 76, 217
Program card, 66
Program execution, 38
Recover from a lost password, 53
Security levels, 52
Signal board (SB), 13
Startup parameters, 38, 63
Startup processing, 40
STOP mode, 276
Thermal zone, 24
Transfer card, 63
Unspecific CPU, 71
watch tables, 273
Wiring guidelines, 32, 34
Creating a network connection, 75
C-Tick approval, 281
CTRL_PWM instruction, 174
CTS, 239
cULus approval, 280
Customer support, 3
Cycle time, 47, 48
Cycle time monitoring, online CPU, 272

D

Data block
Global data block, 53, 86
Instance data block, 53
organization block (OB), 82
Data block (DB), 86
Data handling block (DHB), 86
Data transmission, initiating, 259
Data types, 57
Arrays, 59
DTL, 60
STRING, 59
Date and Time Long data type, 60
Date instructions, 132
DB (data block), 86
DC

Inductive loads, 34
DEC (decrement) instruction, 115
Decode (DECO) instruction, 128
Designing a PLC system, 79, 80
DETACH interrupt instruction, 167
Device configuration, 69, 211
Add modules, 73
Add new device, 70
Configuring the CPU, 72
Configuring the modules, 74
Discover, 71
Ethernet port, 76, 217
Network connection, 75
PROFINET, 76, 217
Diagnostics buffer, 49, 272
Digital I/O status indicators, 270
Digital signal board (SB) specifications, 313
DIN rail, 25
DIS_AIRT alarm interrupt instruction, 172
Discover, 71
Displaying the contents and index (online help), 17
Divide (DIV) instruction, 113
Documentation, 16
Download to device, 220
DTL data type, 60

E

Electromagnetic compatibility (EMC), 281
EN and ENO (power flow), 88
EN_AIRT alarm interrupt instruction, 172
Encode (ENCO) instruction, 128
End conditions, 244
End message character, 244
Environmental conditions, 282
Environments
industrial, 281
Errors
Diagnostic errors, 46
PtP instructions, 266
Time errors, 45
Ethernet
IP address, 76, 217
Network connection, 75
Ethernet communication, 209
Ethernet instructions
TCON, 159
TDISCON, 159
TRCV, 159
TRCV_C, 154
TSEND, 159
TSEND_C, 154
Event execution, 43
Expanding the capabilities of the S7-1200, 12
Expanding the online help window, 17

F

FB (function block), 84
FBD (function block diagram), 88
FC (function), 84
Fill (FILL_BLK) instruction, 121
Floating-point math instruction, 118
Floor (FLOOR) instruction, 124
Flow control, 239
configuration, 239
FM approval, 280
Freeport protocol, 237
Function (FC), 84
Function block (FB)
Initial value, 84
Instance data block, 84
Output parameters, 85

G

General technical specifications), 279
Getting started
Cascading tool tips, 16
Context-sensitive help, 16
Documentation, 16
Information system, 16
Online help, 16
Portal and project views, 15
Rollout help, 16
Tool tips, 16
Global data block, 53, 86
Global library
USS, 177
Guidelines
Grounding, 33
Inductive loads, 34
Installation, 21
Installation procedures, 25
Isolation, 33
Lamp loads, 35
Wiring guidelines, 32, 34

H

Hardware configuration, 69
Add modules, 73
Add new device, 70

Configuring the CPU, 72
Configuring the modules, 74
Discover, 71
Ethernet port, 76, 217
Network connection, 75
PROFINET, 76, 217
Hardware flow control, 239
Help, 16
Displaying the contents and index, 17
Expanding, 17
Printing, 18
Undocking, 17
High-speed counter, 107
High-speed counter (HSC) instruction, 105
HMI
configuring PROFINET communication, 221
HMI devices
Network connection, 75
Overview, 19
Hotline, 3
HSC (high-speed counter), 107
configuration, 110

I

I/O
Addressing, 57
analog status indicators, 270
digital status indicators, 270
Inductive loads, 34
I/O modules
watch tables, 273
Idle line, 241, 243
INC (increment) instruction, 115
Inductive loads, 34
Information system, 16
Displaying the contents and index, 17
Expanding, 17
Printing, 18
Undocking, 17
Input simulators, 319
In-range instruction, 112
Inserting a device
Unspecific CPU, 71
Installation
Clearance, 22
Communication module (CM), 29
CPU, 26
Grounding, 33
Guidelines, 21
Inductive loads, 34
Isolation guidelines, 33
Lamp loads, 35
Mounting dimensions, 24
Overview, 21
Power budget, 22
Signal board (SB), 30
Signal module (SM), 27
STEP 7, 14
Terminal block connector, 31
Thermal zone, 24
TIA Portal, 14
Wiring guidelines, 32, 34
Instance data block, 53
Instructions
absolute value (ABS), 116
add (ADD), 113
AND, 127
bit logic, 93
block move (MOVE_BLK), 119
calendar, 132
ceiling, 124
clock, 134
clock: read local time (RD_LOC_T), 134
clock: read system time (RD_SYS_T), 134
clock: write system time (WR_SYS_T), 134
compare, 111
convert, 123
counter, 102
CTRL_PWM), 174
date, 132
DEC (decrement), 115
decode (DECO), 128
divide (DIV), 113
encode (ENCO), 128
fill (FILL_BLK), 121
floating-point math, 118
floor (FLOOR), 124
GET_ERROR, 154
high-speed counter (HSC), 105
INC (increment), 115
in-range, 112
interrupt: ATTACH, 167
interrupt: CAN_DINT, 170
interrupt: DETACH, 167
interrupt: DIS_AIRT, 172
interrupt: EN_AIRT, 172
interrupt: SRT_DINT, 170
invert (INV), 127
jump (JMP), 126
label, 126
limit, 117
MAX (maximum), 116
MIN (minimum), 116

MOD (modulo), 114
move, 119
multiplex (MUX), 129
multiply (MUL), 113
NEG (negation), 115
negative edge, 97
normalize (NORM), 124
not OK, 113
OK, 113
OR, 127
out-of-range, 112
PID_Compact, 172
PORT_CFG (port configuration), 250
positive edge, 97
RCV_CFG (receive configuration), 253
RCV_PtP (receive Point-to-Point), 262
RCV_RST (receiver reset), 263
RE_TRIGR, 47, 150
reset, 95
return value (RET), 126
rotate left (ROL), 131
rotate right (ROR), 131
round, 123
scale (SCALE_X), 124
select (SEL), 129
SEND_CFG (send configuration), 251
SEND_PTP (send Point-to-Point data), 259
set, 95
SGN_GET (get RS232 signals), 264
SGN_SET (set RS232 signals), 265
shift left (SHL), 130
shift right (SHR), 130
STP (stop PLC scan cycle), 151
string to value: S_CONV, 136
string to value: STRG_VAL, 136
subtract (SUB), 113
swap, 122
T_ADD, 132
T_CONV, 132
T_DIFF, 132
T_SUB, 132
TCON, 159
TDISCON, 159
time, 132
timer, 99
timer: RT (reset timer), 99
timer: TOF (off-delay timer), 99
timer: TON (on-delay timer), 99
timer: TONR (on-delay retentive timer), 99
timer: TP (pulse timer), 99
TRCV, 159
TRCV_C, 154, 229
truncate (TRUNC), 123
TSEND, 159
TSEND_C, 154, 225
uninterruptible fill (UFILL_BLK), 121
uninterruptible move (UMOVE_BLK), 119
USS status codes, 186
USS_DRV, 180
USS_PORT, 182
USS_RPM, 183
USS_WPM, 184
value to string: S_CONV, 136
value to string: VAL_STRG, 136
XOR (exclusive OR), 127

Inter-character gap, 244
Interrupt latency, 45
Interrupts
  Overview, 42
Invert (INV) instruction, 127
IP address, 76, 77, 217, 218
  assigning, 212, 219
  assigning online, 214
  configuring, 76, 217
IP address, setting the online CPU, 271
IP router, 76, 217
Isolation guidelines, 33

J

JMPN instruction, 126
Jump (JMP) instruction, 126

K

Know-how protection, 89

L

Label instruction, 126
LAD (ladder logic), 87
Lamp loads, 35
LED indicators, 248, 269
Length
  message, 245
Length m, 245
Length n, 245
Limit instruction, 117
Linear programming, 80
Lost password, 53

M

MAC address, 76, 217, 233
Maritime approval, 281
Master polling architecture, 247
Math instructions, 113
MAX (maximum) instruction, 116
Maximum message length, 244
MB_COMM_LOAD, 187
MB_MASTER, 189
MB_SLAVE, 199
Memory
  clock memory, 50
  I (process image input), 54
  L (local memory), 53
  load memory, 48
  M (bit memory), 56
  Q (process image output), 55
  retentive memory, 48
  system memory, 50
  Temp memory, 56
  work memory, 48
Memory card
  Configure the startup parameters, 63
  empty transfer card, 53
  Lost password, 53
  Programr card, 66
  Transfer card, 63
Memory card specifications, 318
Memory locations, 53, 54
Memory usage monitoring, online CPU, 272
Message configuration
  instructions, 246
  receive, 242
  transmit, 241
Message end, 244
Message length, 244
Message start, 242
MIN (minimum) instruction, 116
Miscellaneous PtP parameter errors, 268
MOD (modulo) instruction, 114
MODBUS, 187
  MB_Master, 189
  MB_SLAVE, 199
Modules
  Communication module (CM), 14
  Comparison chart, 12
  Configuring parameters, 74
  Signal board (SB), 13
  Signal module (SM), 13
  Thermal zone, 24
Monitoring the program, 91
Mounting
  Clearance, 22
  Communication module (CM), 29
  CPU, 26
  Dimensions, 24
  Grounding, 33
  Guidelines, 21
  Inductive loads, 34
  Isolation, 33
  Lamp loads, 35
  Overview, 25
  Signal board (SB), 30
  Signal module (SM), 27
  Terminal block connector, 31
  Thermal zone, 24
  Wiring guidelines, 32, 34
Move instruction, 119
Multiplex (MUX) instruction, 129
Multiply (MUL) instruction, 113

N

NEG (negation) instruction, 115
Negative edge instruction, 97
Network communication, 210
Network connection, 75
  multiple CPUs, 224
Network time protocol (NTP), 234
Normalize (NORM) instruction, 124
Not OK instruction, 113
Numbers
  Floating point, 58
  Real, 58

O

Off-delay (TOF) instruction, 99
OK instruction, 113
On-delay delay (TON) instruction, 99
On-delay retentive (TONR) instruction, 99
Online CPU, 271
  cycle time monitoring, 272
  memory usage monitoring, 272
  operating panel, 272
Online help, 16
  Displaying the contents and index, 17
  Expanding the help window, 17
  Printing, 18
  Undocking, 17
Online, going online, 270
OR instruction, 127
Organization block

Call, 42
configuring operation, 84
creating, 83
Function, 42
multiple cyclic, 83
Priority classes, 42
processing, 82
Out-of-range instruction, 112
Output parameters, 85

P

Parameter assignment, 85
Parameters configuration
receive, 230
transmit, 226
Parity, 239
Password, 53
Password protection
Code block, 89
CPU, 52
Empty transfer card, 53
lost password, 53
PID_Compact instruction, 172
PLC
Overview, 11
system design, 79
using blocks, 80
Point-to-point communication, 237
Point-to-Point programming, 246
Polling architecture, 247
Port configuration, 238
instructions, 246
Port configuration errors, 266
PORT_CFG (port configuration) instruction, 250
Portal view, 15
Add modules, 73
Add new device, 70
Configuring the CPU, 72
Configuring the Ethernet port, 76, 217
Configuring the modules, 74
PROFINET, 76, 217
Positive edge instruction, 97
Power budget, 22, 321
sample, 322, 323
Power requirements
calculating, 322, 323
Printing the help topics, 18
Priorities in processing, 43
Priority class
Overview, 42
PROFINET, 209
IP address, 76, 217
Network connection, 75
testing a network, 219
PROFINET interface
Ethernet address properties, 77, 218
Time synchronization properties, 235
Program card, 66
Program execution, 37
Program structure, 81
Programming
FBD (function block diagram), 88
LAD (ladder), 87
Linear, 80
power flow (EN and ENO), 88
PtP instructions, 246
Structured, 80
Unspecific CPU, 71
Programy card
Configure the startup parameters, 63
Project
Empty transfer card, 53
Lost password, 53
Programr card, 66
Protecting a code block, 89
Restricting access to a CPU, 52
transfer card, 63
Project view, 15
Add modules, 73
Add new device, 70
Configuring the CPU parameters, 72
Configuring the Ethernet port, 76, 217
Configuring the modules, 74
Device configuration, 69
Network connection, 75
PROFINET, 76, 217
Protection class, 283
Protection level
Code block, 89
CPU, 52
lost password, 53
Protocol
communication, 237
freeport, 237
PTO (pulse train output), 175
PtP communication, 237
PtP instruction return values, 266
PtP programming, 246
Pulse delay (TP) instruction, 99
Pulse train output (PTO), 175
PWM
CTRL_PWM instruction, 174

Q

Queuing, 43

R

Rated voltages, 283
RCV_CFG (receive configuration) instruction, 253
RCV_PTP (receive Point-to-Point) instruction, 262
RCV_RST (receiver reset) instruction, 263
RE_TRIGR instruction, 150
Receive configuration errors, 267
Receive message configuration, 242
Receive parameters configuration, 230
Receive runtime return values, 268
Relay electrical service life, 283
Reset instruction, 95
Reset timer (RT) instruction, 99
Return value (RET) instruction, 126
Return values
  PtP instructions, 266
Rollout help, 16
Rotate left (ROL) instruction, 131
Rotate right (ROR) instruction, 131
Round instruction, 123
Router IP address, 77, 218
RS232 and RS485 communication modules, 237
RT (reset timer) instruction, 99
RTS, 239
RTS always on, 240
RTS Off delay, 241
RTS On delay, 241
RTS switched, 239
RUN mode, 39, 41
RUN to STOP transition, 51

S

S_CONV instruction, 136
S7-1200
  Add modules, 73
  Add new device, 70
  Clearance, 22
  Communication module (CM), 14
  Comparison chart of CPU models, 12
  Configuring the CPU parameters, 72
  Configuring the modules, 74
  CPU, 11
  CPU installation procedures, 26
  Cycle time, 48
  Device configuration, 69
  empty transfer card, 53
  Ethernet port, 76, 217
  Expanding the capabilities, 12
  Grounding, 33
  HMI devices, 19
  Inductive loads, 34
  installing a CM, 29
  installing an SB, 30
  installing an SM, 27
  IP address, 76, 217
  Isolation guidelines, 33
  Lamp loads, 35
  Lost password, 53
  Mounting dimensions, 24
  Network connection, 75
  Overview of installation, 25
  Password protection, 52
  Power budget, 22
  PROFINET, 76, 217
  Program card, 66
  Signal board (SB), 13
  Signal module (SM), 13
  Startup parameters, 38, 63
  Terminal block connector, 31
  Thermal zone, 24
  Transfer card, 63
  Wiring guidelines, 32, 34
SB 1223 specifications, 313, 315
SB 1223 wiring diagram, 314
SB 1232 wiring diagram, 316
Scale (SCALE_X instruction, 124
Scan cycle time, 47
Security
  Code block, 89
  CPU, 52
  Lost password, 53
Select (SEL) instruction, 129
Send message configuration, 241
Send parameters configuration, 226
SEND_CFG (send configuration) instruction, 251
SEND_PtP (send Point-to-Point data) instruction, 259
Serial communication, 237
Set instruction, 95
SGN_GET (get RS232 signals) instruction, 264
SGN_SET (set RS232 signals) instruction, 265
Shift left (SHL) instruction, 130
Shift right (SHR) instruction, 130
Signal board (SB)
  Add modules, 73
  Comparison chart, 12
  Device configuration, 69
  Installation, 30
  Overview, 13

power requirements, 321
Removal, 30
Signal board (SM)
Add new device, 70
Signal handling errors, 267
Signal module (SM)
Add modules, 73
Add new device, 70
Comparison chart, 12
Device configuration, 69
Installation, 27
Overview, 13
power requirements, 321
Removal, 27
Signal modules
SM 1221 specifications, 299
SM 1222 specifications, 301
SM 1223 specifications, 303
Single instance
Example, 85
Slave polling architecture, 247
Software flow control, 240
Specifications
Analog dignal module voltage, 308
Analog signal modules, 306
ATEX approval, 280
CE approval, 279
communication module CM 1241 RS232, 318
communication module CM 1241 RS485, 317
CPU 1211C, 284
CPU 1212C, 289
CPU 1214C, 294
C-Tick approval, 281
cULus approval, 280
digital signal boards (SBs), 313
electromagnetic compatibility (EMC), 281
environmental conditions, 282
environments, 281
FM approval, 280
general technical, 279
input simulators, 319
maritime approval, 281
memory cards, 318
protection, 283
rated voltages, 283
relay electrical service life, 283
SB 1223, 313
SB 1223, 315
SM 1221 signal module, 299
SM 1221 wiring diagram, 300
SM 1222 signal module, 301
SM 1222 wiring diagram, 302
SM 1223 signal module, 303
SM 1223 wiring diagram, 304
wiring diagrams: SM 1231, 1232, 1234, 311
SRT_DINT time delay interrupt instruction, 170
Start conditions, 242
Start message character, 243
Startup parameters, 38, 63
STEP 7
Add modules, 73
Add new device, 70
Configuring the CPU, 72
Configuring the modules, 74
Device configuration, 69
Ethernet port, 76, 217
Installation, 14
Network connection, 75
Portal view, 15
PROFINET, 76, 217
Project view, 15
Stop bits, 239
STOP mode, 39, 276
STP (stop PLC scan cycle) instruction, 151
STRG_VAL instruction, 136
String data type, 59
String to value instructions, 136
Structured programming, 80, 81
Subnet mask, 76, 217
Subtract (SUB) instruction, 113
Support, 3
Swap instruction, 122

T

T_ADD instruction, 132
T_CONV instruction, 132
T_DIFF instruction, 132
T_SUB instruction, 132
TCON instruction, 159
TCP/IP communication, 209
TDISCON instruction, 159
Technical specifications, 279
Technical support, 3
Terminal block connector
Installation, 31
Testing the program, 91
Thermal zone, 24
TIA Portal
Add modules, 73
Add new device, 70
Configuring the CPU, 72
Configuring the modules, 74
Device configuration, 69

Ethernet port, 76, 217
Installation, 14
Network connection, 75
Portal view, 15
PROFINET, 76, 217
Project view, 15
Time instructions, 132
Time of day, setting the online CPU, 271
Timer instructions, 99
TOF (off-delay) timer instruction, 99
TON (on-delay delay) timer instruction, 99
TONR (on-delay retentive) timer instruction, 99
Tool tips, 16
TP (pulse delay) timer instruction, 99
Transfer card, 63
Configure the startup parameters, 63
Empty transfer card, 53
Lost password, 53
Transmission block (T-block), 225
Transmit configuration errors, 267
Transmit message configuration, 241
Transmit runtime errors, 267
TRCV instruction, 159
TRCV_C instruction, 154, 229
TRCV_C instruction configuration, 230
Truncate (TRUNC) instruction, 123
TSAP (transport service access points, 227
TSAP (transport service access points), 231
TSEND instruction, 159
TSEND_C instruction, 154, 225
TSEND_C instruction configuration, 226

U

Undocking the online help, 17
Uninterruptible fill (UFILL_BLK) instruction, 121
Uninterruptible move (UMOVE_BLK) instruction, 119
Unspecific CPU, 71
USS protocol library, 177
USS status codes, 186
USS_DRV instruction, 180
USS_PORT instruction, 182
USS_RPM instruction, 183
USS_WPM instruction, 184

V

VAL_STRG instruction, 136
Value to string instructions, 136

W

Wait time, 239
Watch tables, 91, 273
Watchdog, 150
Wiring diagrams
CPU 1211C, 287
CPU 1212C, 292
CPU 1214C, 297
SB 1223, 314
SB 1232, 316
SM 1221 signal module, 300
SM 1222 signal module, 302
SM 1223 signal module, 304
SM 1231, 1232, 1234, 311
Wiring guidelines
Grounding, 33
Prerequisites, 32

X

XON / XOFF, 240
XOR (exclusive OR) instruction, 127