4.4.9 Pointer data types

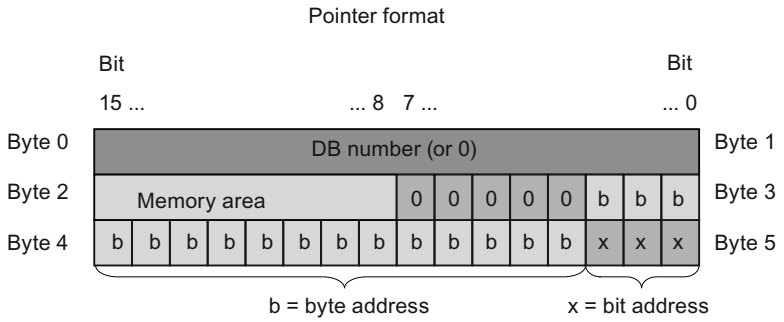
The pointer data types (Pointer, Any, and Variant) can be used in the block interface tables for FB and FC code blocks. You can select a pointer data type from the block interface data type drop-lists.

The Variant data type is also used for instruction parameters.

4.4.9.1 "Pointer" pointer data type

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

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



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

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

Table 4- 26 Pointer types:

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

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

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

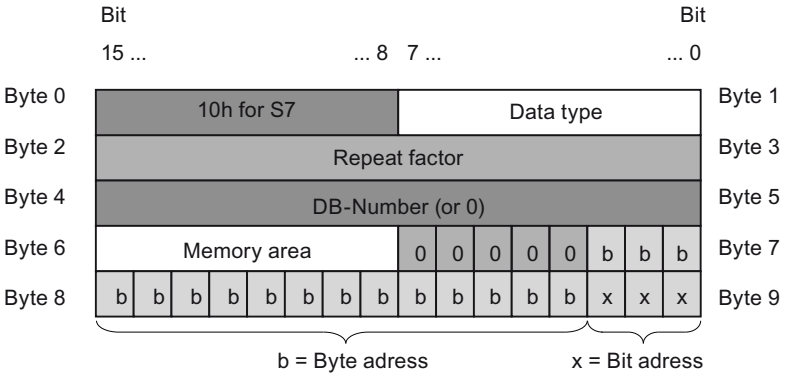
Hexadecimal code	Data type	Description
b#16#81	I	Input memory area
b#16#82	Q	Output memory area
b#16#83	M	Marker memory area
b#16#84	DBX	Data block
b#16#85	DIX	Instance data block
b#16#86	L	Local data
b#16#87	V	Previous local data

4.4.9.2 "Any" pointer data type

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

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

The following image shows the structure of the ANY pointer:



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

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

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

Table 4- 29 Data type encoding in the ANY pointer

Hexadecimal code	Data type	Description
b#16#00	Null	Null pointer
b#16#01	Bool	Bits
b#16#02	Byte	Bytes, 8 Bits
b#16#03	Char	8-bit character
b#16#04	Word	16-bit-word
b#16#05	Int	16-bit-integer
b#16#37	SInt	8-bit-integer
b#16#35	UInt	16-bit unsigned integer
b#16#34	USInt	8-bit unsigned integer
b#16#06	DWord	32-bit double word
b#16#07	DInt	32-bit double integer
b#16#36	UDInt	32-bit-unsigned double integer
b#16#08	Real	32-Bit floating point
b#16#0B	Time	Time
b#16#13	String	Character string

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

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

4.4.9.3 "Variant" pointer data type

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

Table 4- 31 Properties of the Variant pointer

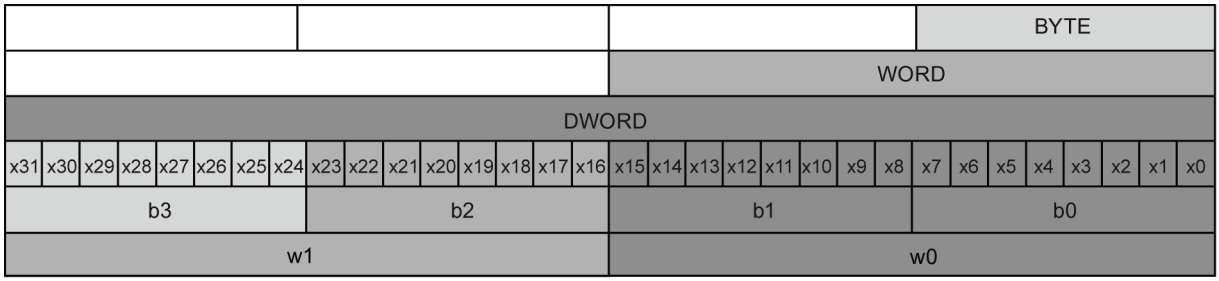
Length (Byte)	Representation	Format	Example entry
0	Symbolic	Operand	MyTag
		DB_name.Struct_name.element_name	MyDB.Struct1.pressure1
	Absolute	Operand	%MW10
		DB_number.Operand Type Length	P#DB10.DBX10.0 INT 12

4.4.10 Accessing a "slice" of a tagged data type

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

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

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



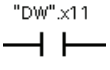
Note

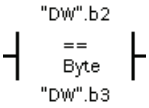
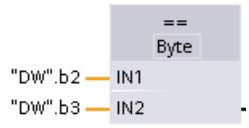
Valid data types that can be accessed by slice are Byte, Char, Conn_Any, Date, DInt, DWord, Event_Any, Event_Att, Hw_Any, Hw_Device, HW_Interface, Hw_Io, Hw_Pwm, Hw_SubModule, Int, OB_Any, OB_Att, OB_Cyclic, OB_Delay, OB_WHINT, OB_PCYCLE, OB_STARTUP, OB_TIMEERROR, OB_Tod, Port, Rtm, SInt, Time, Time_Of_Day,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

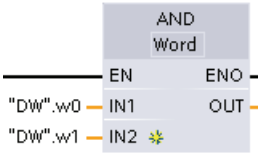
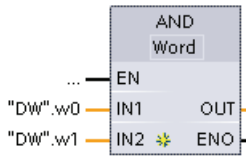
Examples

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

	LAD	FBD	SCL

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

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

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

See also

SCL (Page 156)

4.4.11 Accessing a tag with an AT overlay

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

Declaration

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

Example

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

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

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

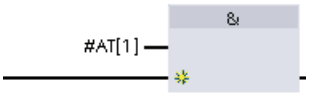
7	6	5	4	3	2	1	0
AT[0]	AT[1]	AT[2]	AT[3]	AT[4]	AT[5]	AT[6]	AT[7]

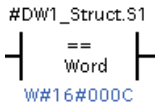
Another example is a DWord tag overlaid with a Struct:

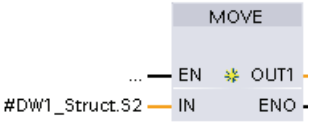
DW1		DWord
DW1_Struct	AT "DW1"	Struct
S1		Word
S2		Byte
S3		Byte

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

LAD	FBD	SCL

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

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

		`out1 := #DW1_Struct.S2;`

Rules

- Overlaying of tags is only possible in FB and FC blocks with standard access.
- You can overlay parameters for all block types and all declaration sections.

- An overlaid parameter can be used like any other block parameter.
- You cannot overlay parameters of type VARIANT.
- The size of the overlaying parameter must be less than or equal to the size of the overlaid parameter.
- The overlaying variable must be declared immediately after the variable that it overlays and identified with the keyword "AT".

See also

SCL (Page 156)

4.5 Using a memory card

NOTICE
The CPU supports only the pre-formatted SIMATIC memory card (Page 826).
Before you copy any program to the formatted memory card, delete any previously saved program from the memory card.

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

- Use a transfer card (Page 110) to copy a program to the internal load memory of the CPU without using STEP 7. After you insert the transfer card, the CPU first erases the user program and any force values from the internal load memory, and then copies the program from the transfer card to the internal load memory. When the transfer process is complete, you must remove the transfer card.

  You can use an empty transfer card to access a password-protected CPU when the password has been lost or forgotten (Page 118). 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 (Page 112) 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 copied 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, STEP 7 then applies the force values only to the external load memory on the program card.

You also use a memory card when downloading firmware updates (Page 115).

4.5.1 Inserting a memory card in the CPU

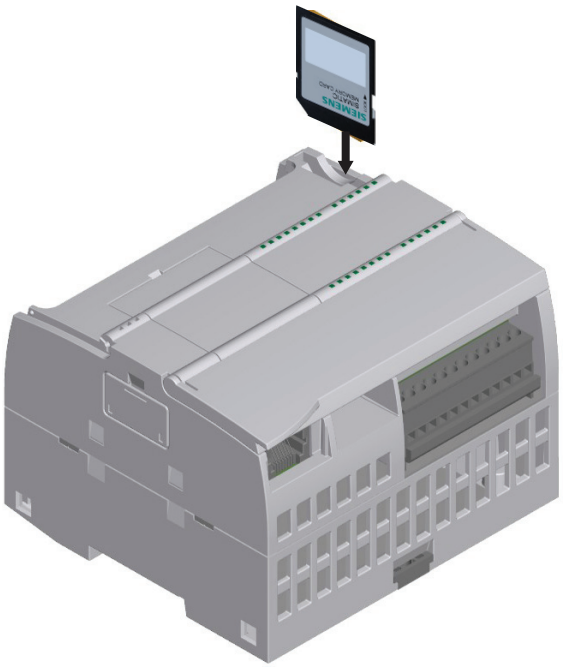
CAUTION
Electrostatic discharge can damage the memory card or the receptacle on the CPU. Make contact with a grounded conductive pad and/or wear a grounded wrist strap when you handle the memory card. Store the memory card in a conductive container.

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

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

Note

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

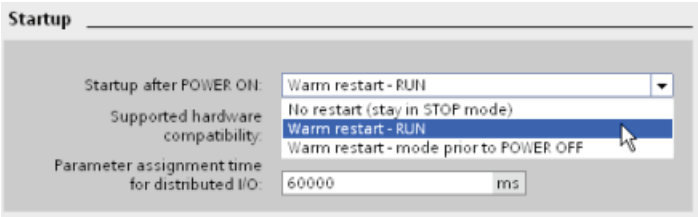


To insert a memory card, open the top CPU door and insert the memory card in the slot. A push-push type connector allows for easy insertion and removal.

The memory card is keyed for proper installation.

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

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



4.5.3 Transfer card

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

Creating a transfer card

Always remember to configure the startup parameter of the CPU (Page 110) before copying a program to the transfer card. To create a transfer card, follow these steps:

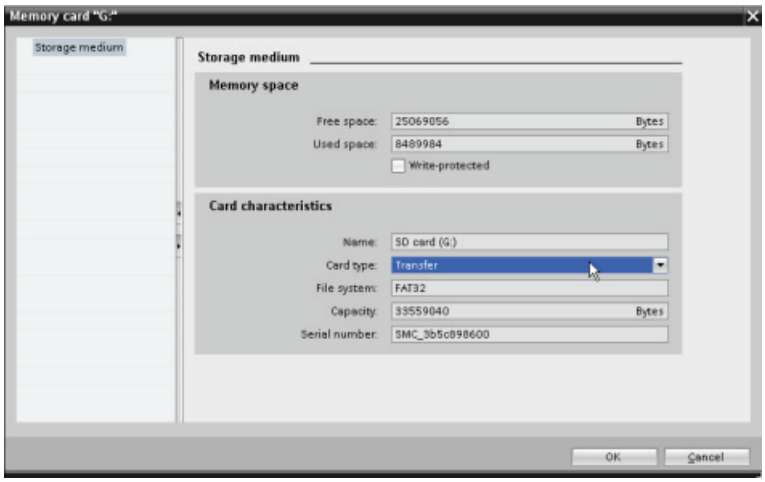
1. Insert a blank SIMATIC memory card into an SD card reader/writer attached to your computer.

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

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

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

   At this point, STEP 7 creates the empty transfer card. If you are creating an empty transfer card, such as to recover from a lost CPU password (Page 118), remove the transfer card from the card reader.



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

Using a transfer card

WARNING

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

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

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

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

1. Insert the transfer card into the CPU (Page 108). If the CPU is in RUN, the CPU will go to STOP mode. The maintenance (MAINT) LED flashes to indicate that the memory card needs to be evaluated.
2. Power-cycle the CPU to evaluate the memory card. Alternative methods for rebooting the CPU are to perform either a STOP-to-RUN transition or a memory reset (MRES) from STEP 7.

3. After the rebooting and evaluating the memory card, the CPU copies the program to the internal load memory of the CPU.

   The RUN/STOP LED alternately flashes green and yellow to indicate that the program is being copied. When the RUN/STOP LED turns on (solid yellow) and the MAINT LED flashes, the copy process has finished. You can then remove the memory card.

4. Reboot the CPU (either by restoring power or by the alternative methods for rebooting) to evaluate the new program that was transferred to internal load memory.

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

Note

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

4.5.4 Program card

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

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

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

Creating a program card

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

Note

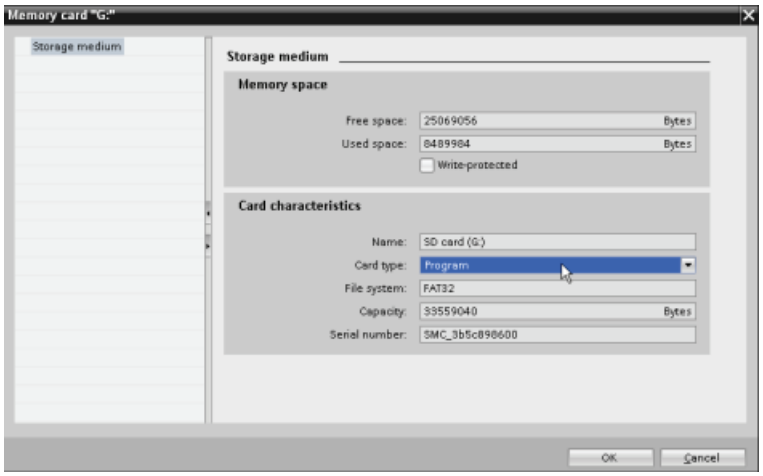
If you insert a blank memory card into the CPU and perform a memory card evaluation by either power cycling the CPU, performing a STOP to RUN transition, or performing a memory reset (MRES), the program and force values in internal load memory of the CPU are copied to the memory card. (The memory card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

Always remember to configure the startup parameter of the CPU (Page 110) before copying a project to the program card. To create a program card, follow these steps:

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

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

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



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

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

Using a program card as the load memory for your CPU

WARNING

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

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

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

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

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

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

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

WARNING

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

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

4.5.5 Firmware update

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.

You use a memory card when downloading firmware updates from customer support (http://www.siemens.com/automation/). From this Web site, navigate to **Automation Technology > Automation Systems > SIMATIC Industrial Automation Systems > PLC > Modular controllers SIMATIC S7 > SIMATIC S7-1200.** From there continue navigating to the specific type of module that you need to update. Under "Support", click the link for "Software Downloads" to proceed.

Alternatively, you can access the S7-1200 downloads Web page (http://support.automation.siemens.com/WW/view/en/34612486/133100) directly.

Note

You cannot update an S7-1200 CPU V2.2 or earlier to S7-1200 V3.0 by firmware update.

CAUTION
Do not use the Windows formatter utility or any other formatting utility to reformat the memory card. If a Siemens memory card is reformatted using the Microsoft Windows formatter utility, then the memory card will no longer be usable by a S7-1200 CPU.

To download the firmware update to your memory card, follow these steps:

1. Insert a blank SIMATIC MC 24 MB memory card into an SD card reader/writer attached to your computer.

   You can reuse a SIMATIC memory card that contains a user program or another firmware update, but you must delete some of the files on the memory card.

CAUTION
Do NOT delete the hidden files "__LOG__" and "crdinfo.bin" from the memory card. The "__LOG__" and "crdinfo.bin" files are required for the memory card. If you delete these files, you cannot use the memory card with the CPU.

   To reuse a memory card, you **must** delete the "S7_JOB.S7S" file and any existing "Data Logs" folders or any folder (such as "SIMATIC.S7S" or "FWUPDATE.S7S") before downloading the firmware update. However, do **not** delete the files "__LOG__" and "crdinfo.bin". (These files are typically hidden and are required.) Use Windows Explorer to display the contents of the memory card and to delete the file and folders.

2. Select the self-extracting file (.exe) for the firmware update that corresponds to your module, and download it to your computer. Double-click the update file, set the file destination path to be the root directory of the SIMATIC memory card, and start the extraction process. After the extraction is complete, the root directory (folder) of the memory card will contain a "FWUPDATE.S7S" directory and the "S7_JOB.S7S" file.

To install the firmware update, follow these steps:

WARNING

Verify that the CPU is not actively running a process before installing the firmware update.

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

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

1. Insert the memory card into the CPU. If the CPU is in RUN mode, the CPU then goes to STOP mode. The maintenance (MAINT) LED flashes to indicate that the memory card needs to be evaluated.
2. Power-cycle the CPU to start the firmware update. Alternative methods for rebooting the CPU are to perform either a STOP-to-RUN transition or a memory reset (MRES) from STEP 7.

   NOTICE

   To complete the firmware upgrade for the module, you must ensure that the external 24 VDC power to the module remains on.

   After the CPU reboots, the firmware update starts. The RUN/STOP LED alternately flashes green and yellow to indicate that the update is being copied. When the RUN/STOP LED turns on (solid yellow) and the MAINT LED flashes, the copy process has finished. You must then remove the memory card.
3. After removing the memory card, reboot the CPU again (either by restoring power or by the alternative methods for rebooting) to load the new firmware.

The user program and hardware configuration are not affected by the firmware update. When the CPU is powered up, the CPU enters the configured start-up state. (If the startup mode for your CPU was configured to "Warm restart - mode before POWER OFF", the CPU will be in STOP mode because the last state of the CPU was STOP.)

Note

Updating multiple modules connected to CPU

If your hardware configuration contains multiple modules that correspond to a single firmware update file on the memory card, the CPU applies the updates to all applicable modules (CM, SM, SB) in configuration order, that is, by increasing order of the module position in Device Configuration in STEP 7.

If you have downloaded multiple firmware updates to the memory card for multiple modules, the CPU applies the updates in the order in which you downloaded them to the memory card.

4.6 Recovery from a lost password

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

For information about the creation and use of an empty transfer card, see the section of transfer cards (Page 110).

WARNING

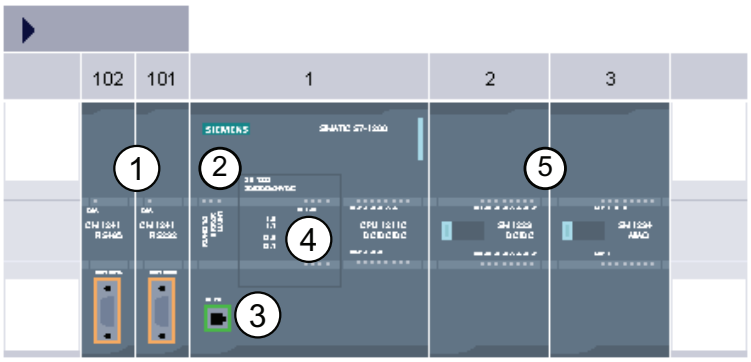
If you insert a transfer card in a running CPU, the CPU goes to STOP. Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment.

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

Device configuration

5

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



① Communication module (CM) or communication processor (CP): Up to 3, inserted in slots 101, 102, and 103

② CPU: Slot 1

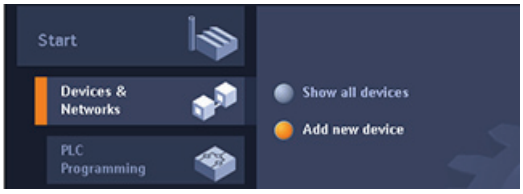
③ Ethernet port of CPU

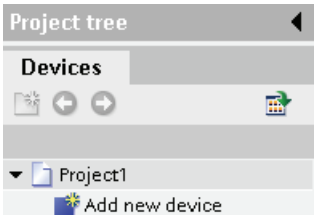
④ Signal board (SB), communication board (CB) or battery board (BB): up to 1, inserted in the CPU

⑤ Signal module (SM) for digital or analog I/O: up to 8, inserted in slots 2 through 9
(CPU 1214C and CPU 1215C allow 8, CPU 1212C allows 2, CPU 1211C does not allow any)

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

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



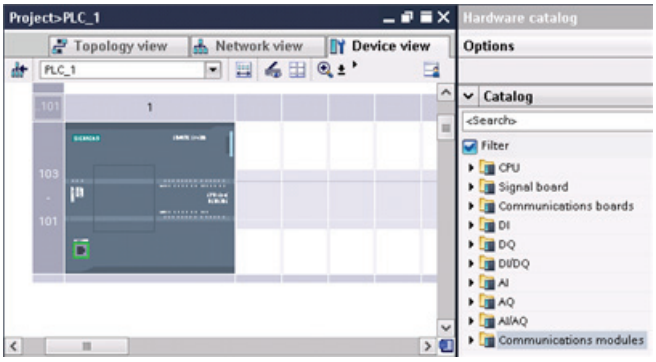


5.1 Inserting a CPU

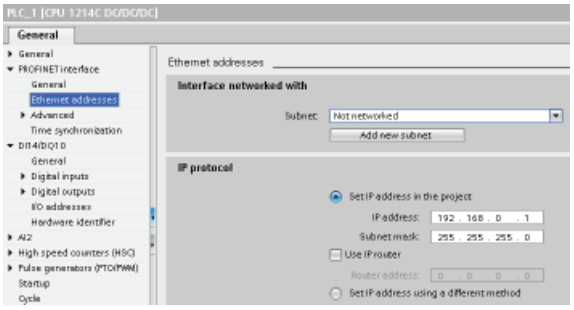
You create your device configuration by inserting a CPU into your project. Be sure you insert the correct model and firmware version from the list. Selecting the CPU from the "Add new device" dialog creates the rack and CPU.

"Add new device" dialog

Device view of the hardware configuration



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



Note

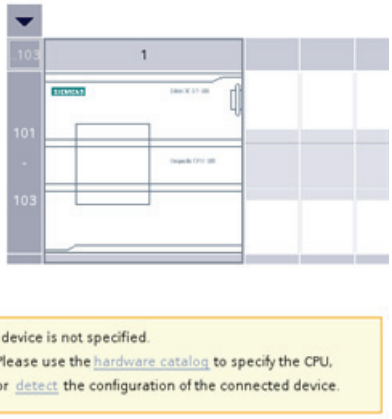
The CPU does not have a pre-configured IP address. You must manually assign an IP address for the CPU during the device configuration. If your CPU is connected to a router on the network, you also enter the IP address for a router.

5.2 Detecting the configuration for an unspecified CPU

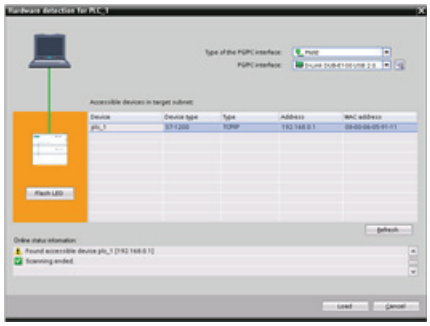
If you are connected to a CPU, you can upload the configuration of that CPU, including any modules, to your project. Simply create a new project and select the "unspecified CPU" instead of selecting a specific CPU. (You can also skip the device configuration entirely by selecting the "Create a PLC program"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

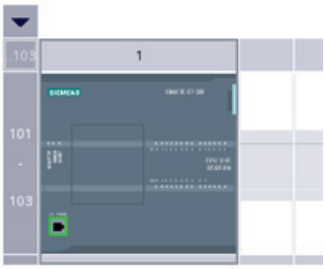
From the program editor, you select the "Hardware detection" command from the "Online" menu.

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



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





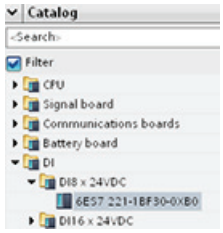
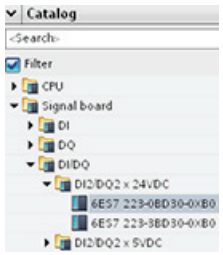
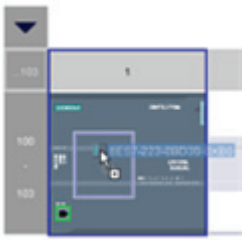
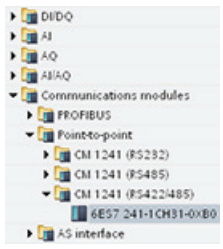
5.3 Adding modules to the configuration

Use the hardware catalog to add modules to the CPU:

- Signal module (SM) provides additional digital or analog I/O points. These modules are connected to the right side of the CPU.
- Signal board (SB) provides just a few additional I/O points for the CPU. The SB is installed on the front of the CPU.
- Battery Board 1297 (BB) provides long-term backup of the realtime clock. The BB is installed on the front of the CPU.
- Communication board (CB) provides an additional communication port (such as RS485). The CB is installed on the front of the CPU.
- Communication module (CM) and communication processor (CP) provide an additional communication port, such as for PROFIBUS or GPRS. These modules are connected to the left side of the CPU.

To insert a module into the device configuration, select the module in the hardware catalog and either double-click or drag the module to the highlighted slot. You must add the modules to the device configuration and download the hardware configuration to the CPU for the modules to be functional.

Table 5- 1 Adding a module to the device configuration

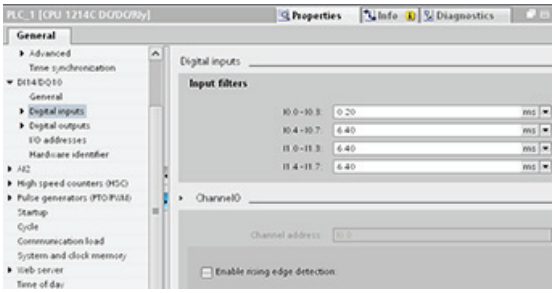
Module	Select the module	Insert the module	Result
SM			
SB, BB or CB			
CM or CP			

5.4 Configuring the operation of the CPU

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

To configure input filter times, select "Digital Inputs". The default filter time for the digital inputs is 6.4 ms.

Each input point has a single filter configuration that applies to all uses: process inputs, interrupts, pulse catch, and HSC inputs.



Note

If an HSC is not configured to use a point, the filter setting chosen in this screen applies. If an HSC is configured to use an input point, the filter setting for that point is automatically set at 800 ns and is not affected by the configuration on this screen.

⚠ WARNING

If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 20.0 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 20.0 ms may not be detected or counted.

This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

To ensure that a new filter time goes immediately into effect, a power cycle of the CPU must be applied.

Table 5- 2 CPU properties

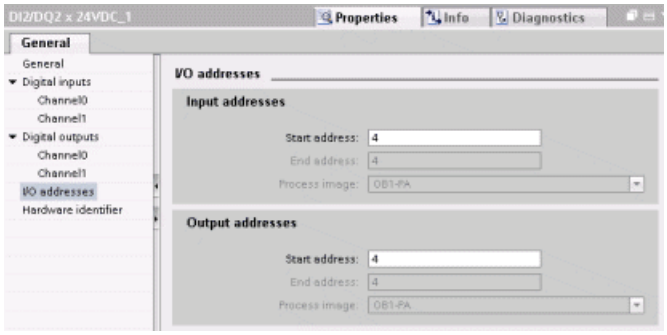
Property	Description
PROFINET interface	Sets the IP address for the CPU and time synchronization
DI, DO, and AI	Configures the behavior of the local (on-board) digital and analog I/O (for example, digital input filter times and digital output reaction to a CPU stop).
High-speed counters (Page 337) and pulse generators (Page 311)	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 motion control instructions), the corresponding output addresses (Q0.0, Q0.1, Q4.0, and Q4.1) are removed from the Q memory and cannot be used for other purposes in your user program. If your user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.

Property	Description
Startup (Page 69)	**Startup after POWER ON:** Selects the behavior of the CPU following an off-to-on transition, such as to start in STOP mode or to go to RUN mode after a warm restart
	Supported hardware compatibility: Configures the substitution strategy for all system components (SM, SB, CM, CP and CPU): • Allow acceptable substitute • Allow any substitute (default) Each module internally contains substitution compatibility requirements based on the number of I/O, electrical compatibility, and other corresponding points of comparison. For example, a 16-channel SM could be an acceptable substitute for an 8-channel SM, but an 8-channel SM could not be an acceptable substitute for a 16-channel SM. If you select "Allow acceptable substitute", STEP 7 enforces the substitution rules; otherwise, STEP 7 allows any substitution.
	Parameter assignment time for distributed I/O: Configures a maximum amount of time (default: 60000 ms) for the distributed I/O to be brought online. (The CMs and CPs receive power and communication parameters from the CPU during startup. This assignment time allows time for the I/O connected to the CM or CP be brought online.) The CPU goes to RUN as soon as the distributed I/O is online, regardless of the assignment time. If the distributed I/O has not been brought online within this time, the CPU still goes to RUN--without the distributed I/O. **Note:** If your configuration uses a CM 1243-5 (PROFIBUS master), do not set this parameter below 15 seconds (15000 ms) to ensure that the module to be brought online.
Cycle (Page 80)	Defines a maximum cycle time or a fixed minimum cycle time
Communication load	Allocates a percentage of the CPU time to be dedicated to communication tasks
System and clock memory (Page 84)	Enables a byte for "system memory" functions and enables a byte for "clock memory" functions (where each bit toggles on and off at a predefined frequency).
Web server (Page 503)	Enables and configures the Web server feature.
Time of day	Selects the time zone and configures daylight saving time
Protection (Page 164)	Sets the read/write protection and password for accessing the CPU
Connection resources (Page 424)	Provides a summary of the communication connections that are available for the CPU and the number of connections that have been configured.
Overview of addresses	Provides a summary of the I/O addresses that have been configured for the CPU.

5.5 Configuring the parameters of the modules

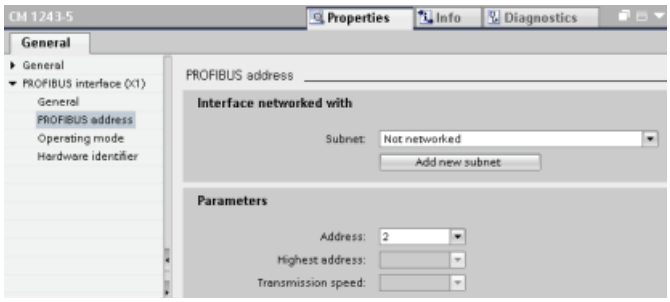
To configure the operational parameters for the modules, select the module in the Device view and use the "Properties" tab of the inspector window to configure the parameters for the module.

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



- Digital I/O: Inputs can be configured for rising-edge detection or falling-edge detection (associating each with an event and hardware interrupt) and also for "pulse catch" (to stay on after a momentary pulse) through the next update of the input process image. Outputs can use a freeze or substitute value.
- Analog I/O: For individual inputs, configure parameters, such as measurement type (voltage or current), range and smoothing, and to enable underflow or overflow diagnostics. Analog outputs provide parameters such as output type (voltage or current) and for diagnostics, such as short-circuit (for voltage outputs) or upper/lower limit diagnostics. You do not configure ranges of analog inputs and outputs in engineering units on the Properties dialog. You must handle this in your program logic as described in the topic "Processing of analog values (Page 92)".
- I/O diagnostic addresses: Configures the start address for the set of inputs and outputs of the module

Configuring a communication interface (CM, CP or CB)



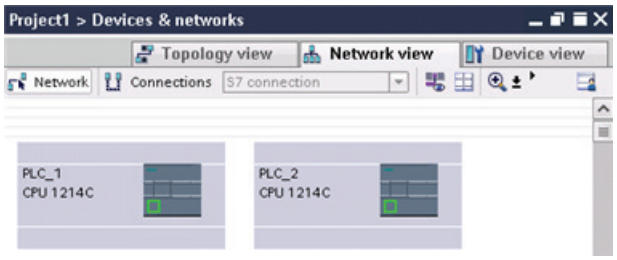
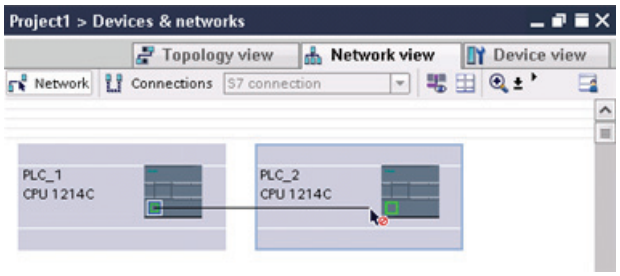
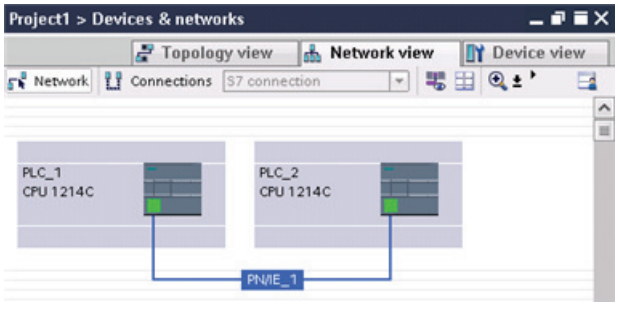
Depending on the type of communication interface, you configure the parameters for the network.

5.6 Configuring the CPU for communication

5.6.1 Creating a network connection

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

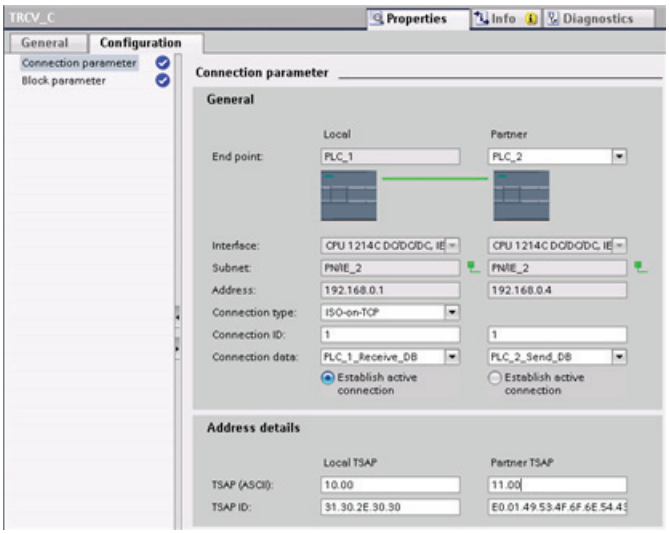
Table 5- 3 Creating a network connection

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

5.6.2 Configuring the Local/Partner connection path

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

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

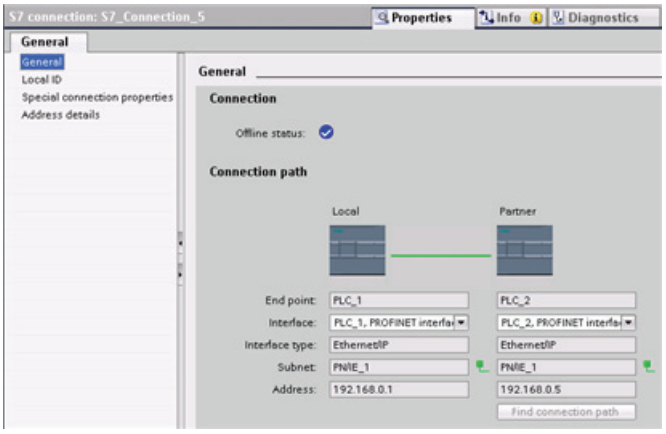
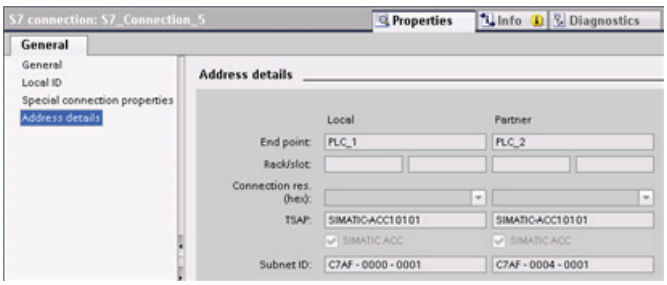
TCP, ISO-on-TCP, and UDP	Connection properties
For the TCP, ISO-on-TCP, and UDP Ethernet protocols, use the "Properties" of the instruction (TSEND_C, TRCV_C, or TCON) to configure the "Local/Partner" connections. The illustration shows the "Connection properties" of the "Configuration tab" for an ISO-on-TCP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

Note

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

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

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

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

Refer to "Protocols" (Page 430) in the "PROFINET" section or to "Creating an S7 connection" (Page 493) in the "S7 communication" section for more information and a list of available communication instructions.

Table 5- 6 Parameters for the multiple CPU connection

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

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

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

Transport Service Access Points (TSAPs)

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

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

Port Numbers

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

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

5.6.3 Parameters for the PROFINET connection

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

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

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

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

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

See also

Configuring the Local/Partner connection path (Page 127)

5.6.4 Assigning Internet Protocol (IP) addresses

5.6.4.1 Assigning IP addresses to programming and network devices

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

Note

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

If your programming device is using an Ethernet-to-USB adapter card connected to an isolated network, the IP Address Network ID and subnet mask of your CPU and the programming device's Ethernet-to-USB adapter card must be exactly the same. The Network ID is the first part of the IP address (first three octets) (for example, **211.154.184**.16) that determines what IP network you are on. The subnet mask normally has a value of **255.255.255.0**. The subnet mask, when combined with the device IP address in a mathematical AND operation, defines the boundaries of an IP subnet.

Note

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

Table 5- 8 Assigning Ethernet addresses

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

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

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

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

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

Note

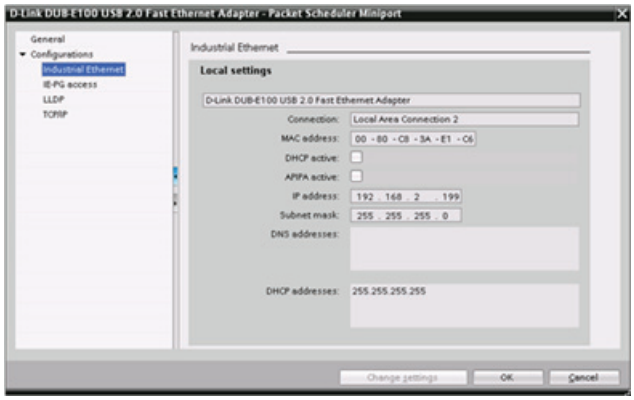
Dynamic Host Configuration Protocol (DHCP) automatically assigns an IP address to your programming device upon power up from the DHCP server.

5.6.4.2 Checking the IP address of your programming device

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

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

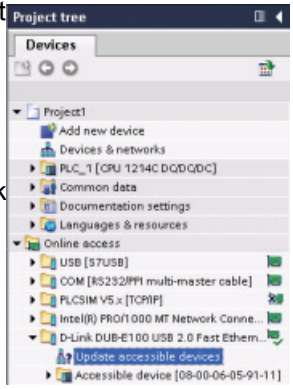
The MAC and IP addresses of the programming device are displayed.



5.6.4.3 Assigning an IP address to a CPU online

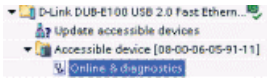
You can assign an IP address to a network device online. This is particularly useful in an initial device configuration.

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



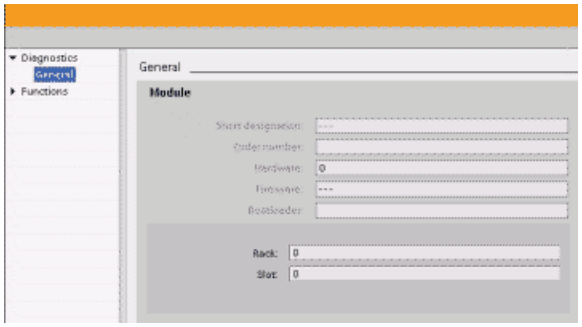
NOTE: If a MAC address is shown instead of an IP address, then no IP address has been assigned.

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

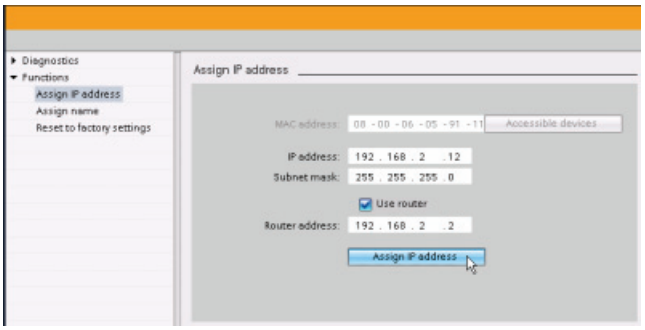


3. In the "Online & diagnostics" dialog, make the following menu selections:

- "Functions"
- "Assign IP address"

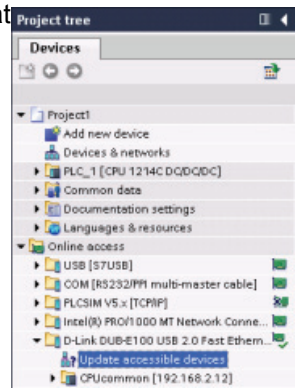


4. In the "IP address" field, enter your new IP address, and click the "Assign IP address" button.



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

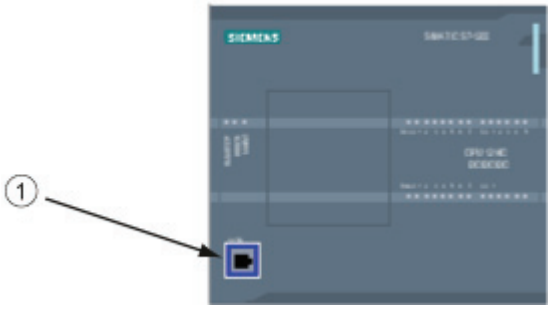
- "Online access"
- <Adapter for the network in which the device is located>
- "Update accessible devices"



5.6.4.4 Configuring an IP address for a CPU in your project

Configuring the PROFINET interface

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



① PROFINET port

Configuring the IP address

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

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

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

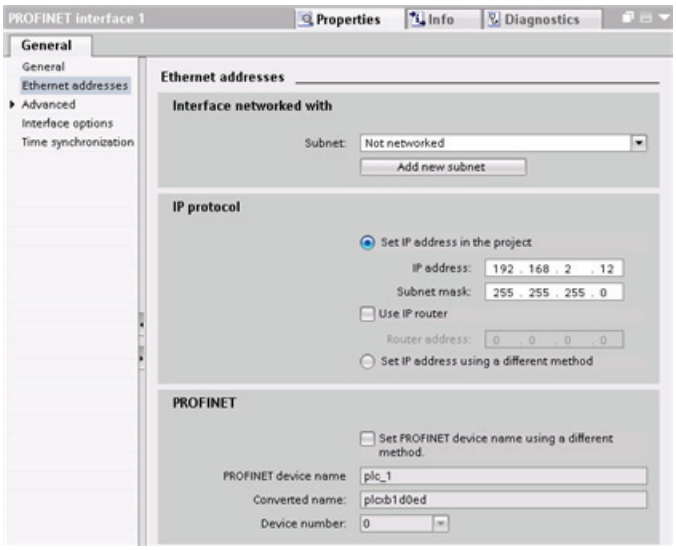
Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet tend to be located in close physical proximity to each other on a Local Area Network (LAN). A mask (known as the subnet mask or network mask) defines the boundaries of an IP subnet.

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

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

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

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



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

Table 5- 9 Parameters for the IP address

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

Note

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

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

WARNING

After downloading a hardware configuration with the "Set IP address using a different method" option enabled, it is not possible to transition the CPU operating mode from RUN to STOP or from STOP to RUN.

User equipment will keep running under these conditions and may result in unexpected machine or process operations, which could cause death, severe personal injury, or property damage if proper precautions are not taken.

Ensure that your CPU IP address(es) are set before using the CPU in an actual automation environment. This can be done by using your STEP 7 programming package, the S7-1200 Tool, or an attached HMI device in conjunction with the T_CONFIG instruction.

WARNING

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

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

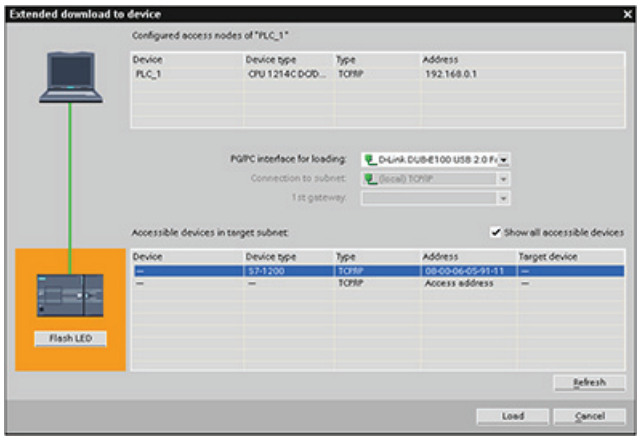
If an IP address must be changed manually, ensure that the new IP address lies within the subnet.

See also

T_CONFIG (Page 451)

5.6.5 Testing the PROFINET network

After completing the configuration, download the project (Page 168) to the CPU. All IP addresses are configured when you download the project.



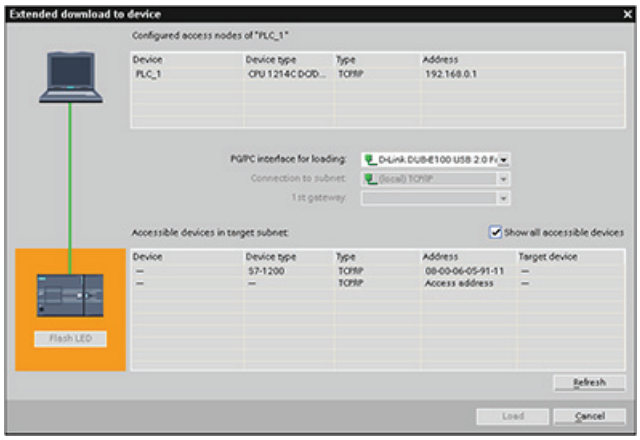
Assigning an IP address to a device online

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

- To assign an IP address to a device online, refer to "Device configuration: Assigning an IP address to a CPU online" (Page 134) for this step-by-step procedure.
- To assign an IP address in your project, you must configure the IP address in the Device configuration, save the configuration, and download it to the PLC. Refer to "Device configuration: Configuring an IP address for a CPU in your project" (Page 136) for more information.

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

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



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

5.6.6 Locating the Ethernet (MAC) address on the CPU

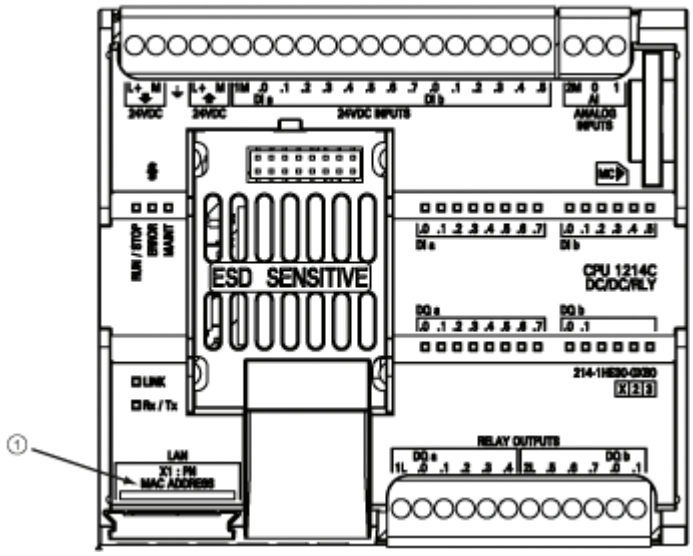
In PROFINET networking, a Media Access Control address (MAC address) is an identifier assigned to the network interface by the manufacturer for identification. A MAC address usually encodes the manufacturer's registered identification number.

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

Note

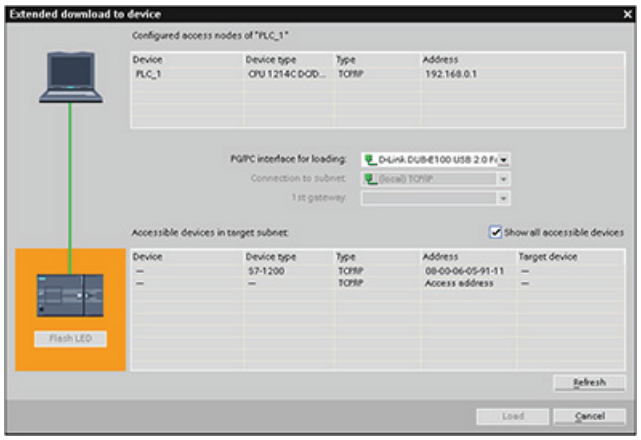
Each CPU is loaded at the factory with a permanent, unique MAC address. You cannot change the MAC address of a CPU.

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



① MAC address

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



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

5.6.7 Configuring Network Time Protocol synchronization

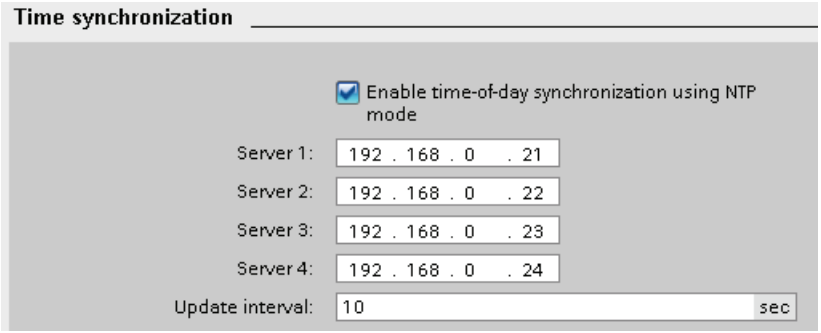
The Network Time Protocol (NTP) is widely used to synchronize the clocks of computer systems to Internet time servers. In NTP mode, the CP sends time-of-day queries at regular intervals (in the client mode) to the NTP server in the subnet (LAN). Based on the replies from the server, the most reliable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

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

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

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

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



Note

All IP addresses are configured when you download the project.

Table 5- 10 Parameters for time synchronization

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

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

PROFINET IO can extend the start-up time for your system (configurable time-out figure). More devices and slow devices impact the amount of time it takes to switch to RUN.

You can have the following maximum numbers of PROFINET IO devices on your S7-1200 PROFINET network:

- In V3.0, you can have a maximum of 16 IO devices.
- In V2.2, you can have a maximum of 8 IO devices.

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

In the CPU Properties under "Startup", you can find the "Parameter assignment time for distributed I/O" (time-out). The default configurable time-out is 60,000 ms (1 minute); the user can configure this time.

PROFINET device naming and addressing in STEP 7

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

PROFINET address assignment at system start-up

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

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

Programming concepts

6

6.1 Guidelines for designing a PLC system

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

Table 6- 1 Guidelines for designing a PLC system

Recommended steps	Tasks
Partition your process or machine	Divide your process or machine into sections that have a level of independence from each other. These partitions determine the boundaries between controllers and influence the functional description specifications and the assignment of resources.
Create the functional specifications	Write the descriptions of operation for each section of the process or machine, such as the I/O points, the functional description of the operation, the states that must be achieved before allowing action for each actuator (such as a solenoid, a motor, or a drive), a description of the operator interface, and any interfaces with other sections of the process or machine.
Design the safety circuits	Identify any equipment that might require hard-wired logic for safety. Remember that control devices can fail in an unsafe manner, which can produce unexpected startup or change in the operation of machinery. Where unexpected or incorrect operation of the machinery could result in physical injury to people or significant property damage, consider the implementation of electromechanical overrides (which operate independently of the PLC) to prevent unsafe operations. The following tasks should be included in the design of safety circuits: • Identify any improper or unexpected operation of actuators that could be hazardous. • Identify the conditions that would assure the operation is not hazardous, and determine how to detect these conditions independently of the PLC. • Identify how the PLC affects the process when power is applied and removed, and also identify how and when errors are detected. Use this information only for designing the normal and expected abnormal operation. You should not rely on this "best case" scenario for safety purposes. • Design the manual or electromechanical safety overrides that block the hazardous operation independent of the PLC. • Provide the appropriate status information from the independent circuits to the PLC so that the program and any operator interfaces have necessary information. • Identify any other safety-related requirements for safe operation of the process.
Plan system security	Determine what level of protection (Page 164) you require for access to your process. You can password-protect CPUs and program blocks from unauthorized access.

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

6.2 Structuring your user program

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

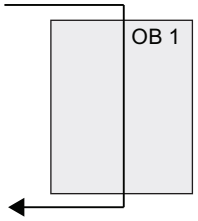
- An organization block (OB) responds to a specific event in the CPU and can interrupt the execution of the user program. The default for the cyclic execution of the user program (OB 1) provides the base structure for your user program and is the only code block required for a user program. If you include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

Choosing the type of structure for your user program

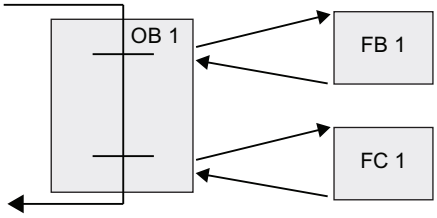
Based on the requirements of your application, you can choose either a linear structure or a modular structure for creating your user program:

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

Linear structure:



Modular structure:



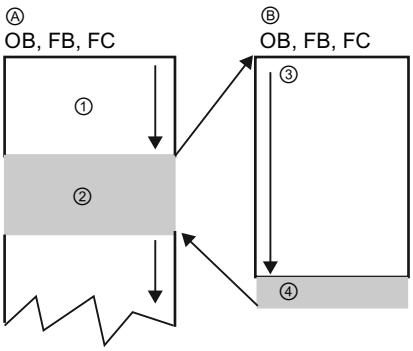
By creating generic code blocks that can be reused within the user program, you can simplify the design and implementation of the user program. Using generic code blocks has a number of benefits:

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

6.3 Using blocks to structure your program

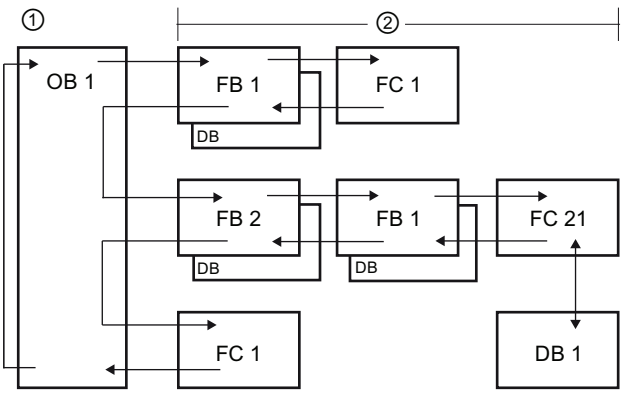
By designing FBs and FCs to perform generic tasks, you create modular code blocks. You then structure your program by having other code blocks call these reusable modules. The calling block passes device-specific parameters to the called block.

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



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

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



① Start of cycle
② Nesting depth

6.3.1 Organization block (OB)

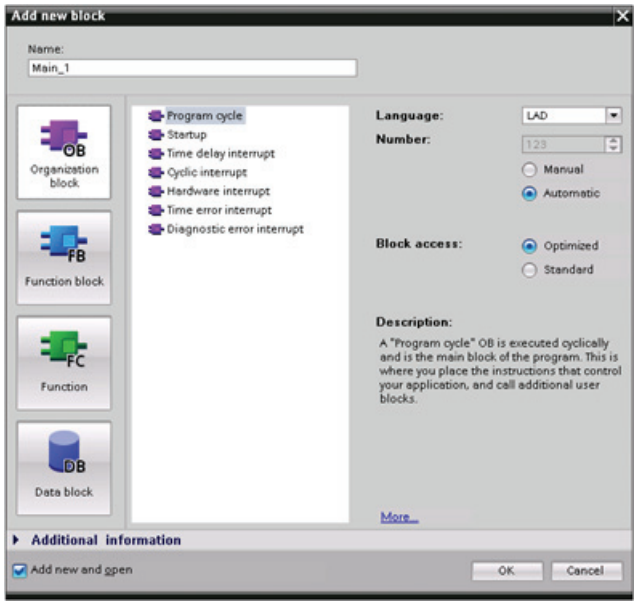
Organization blocks provide structure for your program. They serve as the interface between the operating system and the user program. OBs are event driven. An event, such as a diagnostic interrupt or a time interval, will cause the CPU to execute an OB. Some OBs have predefined start events and behavior.

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

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

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

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



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

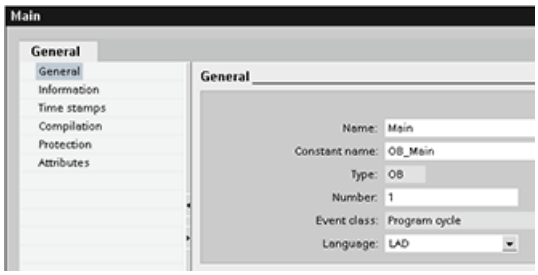
The CPU determines the order for handling interrupt events by a priority assigned to each OB. Each event has a particular servicing priority. The respective priority level within a priority class determines the order in which the OBs are executed. Several interrupt events can be combined into priority classes. For more information, refer to the PLC concepts chapter section on execution of the user program (Page 67).

Creating an additional OB within a class of OB

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

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

Configuring the operation of an OB



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

6.3.2 Function (FC)

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

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

6.3.3 Function block (FB)

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

Reusable code blocks with associated memory

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

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

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

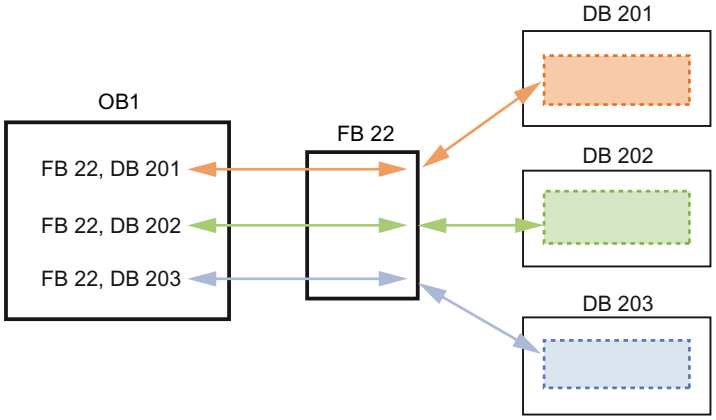
Assigning the start value in the instance DB

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

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

Using a single FB with DBs

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



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

6.3.4 Data block (DB)

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

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

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

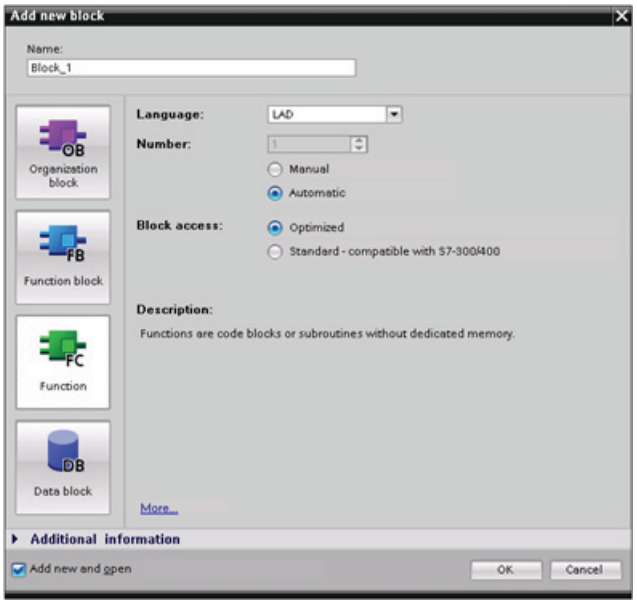
Note

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

You can configure a DB as being read-only:

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

Creating reusable code blocks



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

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

6.4 Understanding data consistency

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

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

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

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

The PtP (Point-to-Point) instructions, PROFINET instructions (such as TSEND_C and TRCV_C), PROFINET Distributed I/O instructions , and PROFIBUS Distributed I/O Instructions (Page 274) transfer buffers of data that could be interrupted. Ensure the data consistency for the buffers of data by avoiding any read or write operation to the buffers in both the program cycle OB and an interrupt OB. If it is necessary to modify the buffer values for these instructions in an interrupt OB, use a DIS_AIRT instruction to delay any interruption (an interrupt OB or a communication interrupt from an HMI or another CPU) until an EN_AIRT instruction is executed.

Note

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

6.5 Programming language

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

- LAD (ladder logic) is a graphical programming language. The representation is based on circuit diagrams (Page 155).
- FBD (Function Block Diagram) is a programming language that is based on the graphical logic symbols used in Boolean algebra (Page 156).
- SCL (structured control language) is a text-based, high-level programming language (Page 156).

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

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

6.5.1 Ladder logic (LAD)

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

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

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

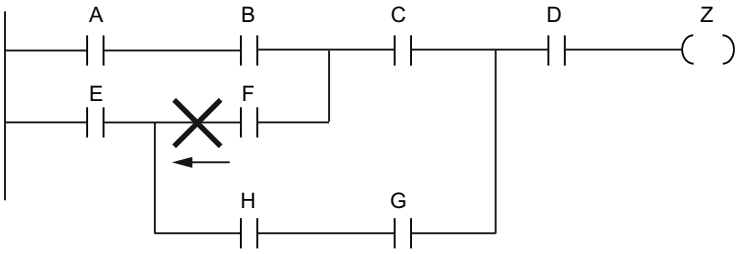
STEP 7 does not limit the number of instructions (rows and columns) in a LAD network.

Note

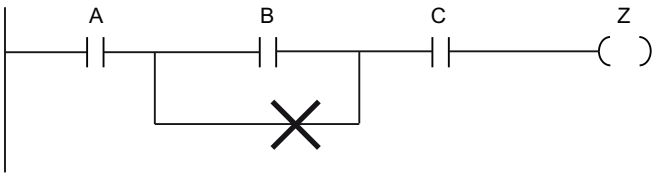
Every LAD network must terminate with a coil or a box instruction.

Consider the following rules when creating a LAD network:

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

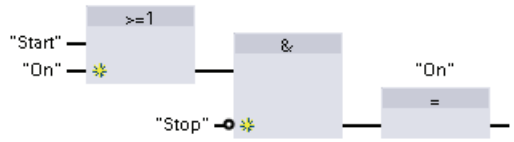


- You cannot create a branch that would cause a short circuit.



6.5.2 Function Block Diagram (FBD)

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



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

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

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

6.5.3 SCL

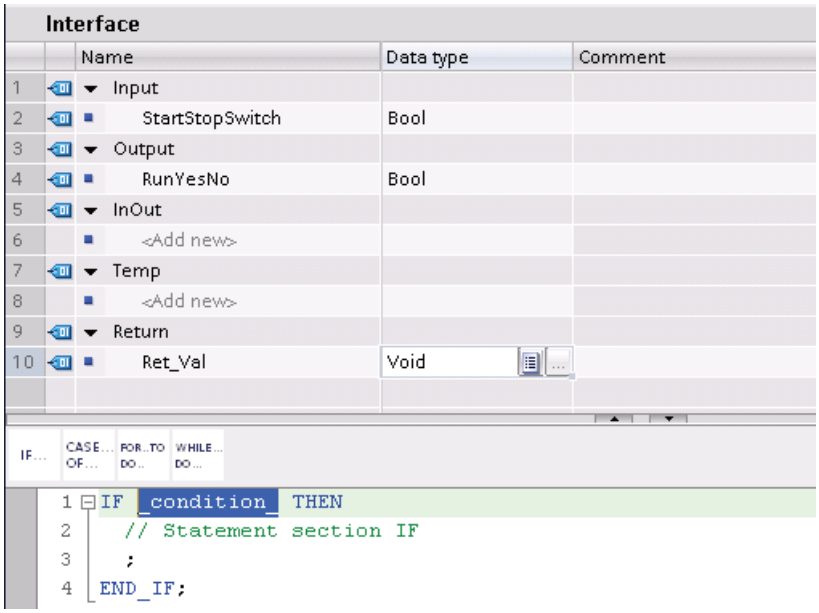
Structured Control Language (SCL) is a high-level, PASCAL-based programming language for the SIMATIC S7 CPUs. SCL supports the block structure of STEP 7 (Page 148). You can also include program blocks written in SCL with program blocks written in LAD and FBD.

SCL instructions use standard programming operators, such as for assignment (:=), mathematical functions (+ for addition, - for subtraction, * for multiplication, and / for division). SCL also uses standard PASCAL program control operations, such as IF-THEN-ELSE, CASE, REPEAT-UNTIL, GOTO and RETURN. You can use any PASCAL reference for syntactical elements of the SCL programming language. Many of the other instructions for SCL, such as timers and counters, match the LAD and FBD instructions. For more information about specific instructions, refer to the specific instructions in the chapters for Basic instructions (Page 175) and Extended instructions (Page 247).

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

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

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



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

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

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

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

Constructing an SCL expression

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

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

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

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

Table 6- 2 Operators in SCL

Type	Operation	Operator	Priority
Parentheses	(*Expression*)	(,)	1
Math	Power	**	2
	Sign (unary plus)	+	3
	Sign (unary minus)	-	3
	Multiplication	*	4
	Division	/	4
	Modulo	MOD	4
	Addition	+	5
	Subtraction	-	5
Comparison	Less than	<	6
	Less than or equal to	<=	6
	Greater than	>	6
	Greater than or equal to	>=	6
	Equal to	=	7
	Not equal to	<>	7
Bit logic	Negation (unary)	NOT	3
	AND logic operation	AND or &	8
	Exclusive OR logic operation	XOR	9
	OR logic operation	OR	10
Assignment	Assignment	:=	11

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

- Assignment statement: :=
- Mathematical functions: +, -, *, and /
- Addressing of global variables (tags): "<tag name>" (Tag name or data block name enclosed in double quotes)
- Addressing of local variables: #<variable name> (Variable name preceded by "#" symbol)

The following examples show different expressions for different uses.

`"C" := #A+#B;`	Assigns the sum of two local variables to a tag
`"Data_block_1".Tag := #A;`	Assignment to a data block tag
`IF #A > #B THEN "C" := #A;`	Condition for the IF-THEN statement
`"C" := SQRT (SQR (#A) + SQR (#B));`	Parameters for the SQRT instruction

Arithmetic operators can process various numeric data types. The data type of the result is determined by the data type of the most-significant operands. For example, a multiplication operation that uses an INT operand and a REAL operand yields a REAL value for the result.

Control statements

A control statement is a specialized type of SCL expression that performs the following tasks:

- Program branching
- Repeating sections of the SCL program code
- Jumping to other parts of the SCL program
- Conditional execution

The SCL control statements include IF-THEN, CASE-OF, FOR-TO-DO, WHILE-DO, REPEAT-UNTIL, CONTINUE, GOTO, and RETURN.

A single statement typically occupies one line of code. You can enter multiple statements on one line, or you can break a statement into several lines of code to make the code easier to read. Separators (such as tabs, line breaks and extra spaces) are ignored during the syntax check. An END statement terminates the control statement.

The following examples show a FOR-TO-DO control statement. (Both forms of coding are syntactically valid.)

```
FOR x := 0 TO max DO sum := sum + value(x); END_FOR;
FOR x := 0 TO max DO
     sum := sum + value(x);
END_FOR;
```

A control statement can also be provided with a label. A label is set off by a colon at the beginning of the statement:

```
Label: <Statement>;
```

The STEP 7 online help provides a complete SCL programming language reference.

Conditions

A condition is a comparison expression or a logical expression whose result is of type BOOL (with the value of either TRUE or FALSE). The following example shows conditions of various types.

`#Temperature > 50`	Relational expression
`#Counter <= 100`	
`#CHAR1 < 'S'`	
`(#Alpha <> 12) AND NOT #Beta`	Comparison and logical expression
`5 + #Alpha`	Arithmetic expression

A condition can use arithmetic expressions:

- The condition of the expression is TRUE if the result is any value other than zero.
- The condition of the expression is FALSE if the result equals zero.

Addressing

As with LAD and FBD, SCL allows you to use either tags (symbolic addressing) or absolute addresses in your user program. SCL also allows you to use a variable as an array index.

Absolute addressing

```
I0.0
MB100
```

Symbolic addressing

`"PLC_Tag_1"`	Tag in PLC tag table
`"Data_block_1".Tag_1`	Tag in a data block
`"Data_block_1".MyArray[#i]`	Array element in a data block array

Indexed addressing with PEEK and POKE instructions

SCL provides PEEK and POKE instructions that allow you to read from or write to data blocks, I/O, or memory. You provide parameters for specific byte offsets or bit offsets for the operation.

Note

To use the PEEK and POKE instructions with data blocks, you must use standard (not optimized) data blocks. Also note that the PEEK and POKE instructions merely transfer data. They have no knowledge of data types at the addresses.

`PEEK(area:=_in_,`
`dbNumber:=_in_,`
`byteOffset:=_in_);`
	Reads the byte referenced by byteOffset of the referenced data block, I/O or memory area.
	Example referencing data block:
	`%MB100 := PEEK(area:=16#84, dbNumber:=1, byteOffset:=#i);`
	Example referencing IB3 input:
	`%MB100 := PEEK(area:=16#81, dbNumber:=0, byteOffset:=#i); // when #i = 3`

`PEEK_WORD(area:=_in_,`
`dbNumber:=_in_,`
`byteOffset:=_in_);`
	Reads the word referenced by byteOffset of the referenced data block, I/O or memory area.
	Example:
	`%MW200 := PEEK_WORD(area:=16#84, dbNumber:=1, byteOffset:=#i);`

`PEEK_DWORD(area:=_in_,`
`dbNumber:=_in_,`
`byteOffset:=_in_);`
	Reads the double word referenced by byteOffset of the referenced data block, I/O or memory area.
	Example:
	`%MD300 := PEEK_DWORD(area:=16#84, dbNumber:=1, byteOffset:=#i);`

`PEEK_BOOL(area:=_in_,`
`dbNumber:=_in_,`
`byteOffset:=_in_,`
`bitOffset:=_in_);`
	Reads a Boolean referenced by the bitOffset and byteOffset of the referenced data block, I/O or memory area
	Example:
	`%MB100.0 := PEEK_BOOL(area:=16#84, dbNumber:=1, byteOffset:=#ii, bitOffset:=#j);`

`POKE(area:=_in_,`
`dbNumber:=_in_,`
`byteOffset:=_in_,`
`value:=_in_);`
	Writes the value (Byte, Word, or DWord) to the referenced byteOffset of the referenced data block, I/O or memory area
	Example referencing data block:
	`POKE(area:=16#84, dbNumber:=2, byteOffset:=3, value:="Tag_1");`
	Example referencing QB3 output:
	`POKE(area:=16#82, dbNumber:=0, byteOffset:=3, value:="Tag_1");`

```
POKE_BOOL(area:=_in_,
          dbNumber:=_in_,
          byteOffset:=_in_,
          bitOffset:=_in_,
          value:=_in_);
```

Writes the Boolean value to the referenced bitOffset and byteOffset of the referenced data block, I/O or memory area

Example:

```
POKE_BOOL(area:=16#84, dbNumber:=2,
byteOffset:=3, bitOffset:=5,
value:=0);
```

```
POKE_BLK(area_src:=_in_,
          dbNumber_src:=_in_,
          byteOffset_src:=_in_,
          area_dest:=_in_,
          dbNumber_dest:=_in_,
          byteOffset_dest:=_in_,
          count:=_in_);
```

Writes "count" number of bytes starting at the referenced byte Offset of the referenced source data block, I/O or memory area to the referenced byteOffset of the referenced destination data block, I/O or memory area

Example:

```
POKE_BLK(area_src:=16#84,
dbNumber_src:=#src_db,
byteOffset_src:=#src_byte,
area_dest:=16#84,
dbNumber_dest:=#src_db,
byteOffset_dest:=#src_byte,
count:=10);
```

For PEEK and POKE instructions, the following values for the "area", "area_src" and "area_dest" parameters are applicable. For areas other than data blocks, the dbNumber parameter must be 0.

16#81	I
16#82	Q
16#83	M
16#84	DB

Calling other code blocks from your SCL program

To call another code block in your user program, simply enter the name (or absolute address) of the FB or FC with the parameters. For an FB, you must provide the instance DB to be called with the FB.

```
<DB name> (Parameter list)
```

Call as a single instance

```
<#Instance name> (Parameter list)
```

Call as multi-instance

```
"MyDB"(MyInput:=10, MyInOut:="Tag1");
```

```
<FC name> (Parameter list)
```

Standard call

```
<Operand>:=<FC name> (Parameter list)
```

Call in an expression

```
"MyFC"(MyInput:=10, MyInOut:="Tag1");
```

You can also drag blocks from the navigation tree to the SCL program editor, and complete the parameter assignment.

6.5.4 EN and ENO for LAD, FBD and SCL

Determining "power flow" (EN and ENO) for an instruction

Certain instructions (such as the Math and the Move instructions) provide parameters for EN and ENO. These parameters relate to power flow in LAD or FBD and determine whether the instruction is executed during that scan. SCL also allows you to set the ENO parameter for a code block.

- EN (Enable In) is a Boolean input. Power flow (EN = 1) must be present at this input for the box instruction to be executed. If the EN input of a LAD box is connected directly to the left power rail, the instruction will always be executed.
- ENO (Enable Out) is a Boolean output. If the box has power flow at the EN input and the box executes its function without error, then the ENO output passes power flow (ENO = 1) to the next element. If an error is detected in the execution of the box instruction, then power flow is terminated (ENO = 0) at the box instruction that generated the error.

Table 6- 3 Operands for EN and ENO

Program editor	Inputs/outputs	Operands	Data type
LAD	EN, ENO	Power flow	Bool
FBD	EN	I, I:P, Q, M, DB, Temp, Power Flow	Bool
	ENO	Power Flow	Bool
SCL	EN[1]	TRUE, FALSE	Bool
	ENO[2]	TRUE, FALSE	Bool

1 The use of EN is only available for FBs.

2. The use of ENO with the SCL code block is optional. You must configure the SCL compiler to set ENO when the code block finishes.

Configuring SCL to set ENO

To configure the SCL compiler for setting ENO, follow these steps:

1. Select the "Settings" command from the "Options" menu.

2. Expand the "PLC programming" properties and select "SCL (Structured Control Language)".

3. Select the "Set ENO automatically" option.

Effect of Ret_Val or Status parameters on ENO

Some instructions, such as the communication instructions or the string conversion instructions, provide an output parameter that contains information about the processing of the instruction. For example, some instructions provide a Ret_Val (return value) parameter, which is typically an Int data type that contains status information in a range from -32768 to +32767. Other instructions provide a Status parameter, which is typically a Word data type that stores status information in a range of hexadecimal values from 16#0000 to 16#FFFF. The numerical value stored in a Ret_Val or a Status parameter determines the state of ENO for that instruction.

- Ret_Val: A value from 0 to 32767 typically sets ENO = 1 (or TRUE). A value from -32768 to -1 typically sets ENO = 0 (or FALSE). To evaluate Ret_Val, change the representation to hexadecimal.
- Status: A value from 16#0000 16#7FFF typically sets ENO = 1 (or TRUE). A value from 16#8000 to 16#FFFF typically sets ENO = 0 (or FALSE).

Instructions that take more than one scan to execute often provide a Busy parameter (Bool) to signal that the instruction is active but has not completed execution. These instructions often also provide a Done parameter (Bool) and an Error parameter (Bool). Done signals that the instruction was completed without error, and Error signals that the instruction was completed with an error condition.

- When Busy = 1 (or TRUE), ENO = 1 (or TRUE).
- When Done = 1 (or TRUE), ENO = 1 (or TRUE).
- When Error = 1 (or TRUE), ENO = 0 (or FALSE).

See also

OK and Not OK instructions (Page 197)

6.6 Protection

6.6.1 Access protection for the CPU

The CPU provides three levels of security for restricting access to specific functions. When you configure the security level and password for a CPU, you limit the functions and memory areas that can be accessed without entering a password.

The password is case-sensitive.

To configure the password, follow these steps:

1. In the "Device configuration", select the CPU.
2. In the inspector window, select the "Properties" tab.
3. Select the "Protection" property to select the protection level and to enter a password.

Each level allows certain functions to be accessible without a password. The default condition for the CPU is to have no restriction and no password-protection. To restrict access to a CPU, you configure the properties of the CPU and enter the password.

Entering the password over a network does not compromise the password protection for the CPU. Password protection does not apply to the execution of user program instructions including communication functions. Entering the correct password provides access to all of the functions.

PLC-to-PLC communications (using communication instructions in the code blocks) are not restricted by the security level in the CPU. HMI functionality is also not restricted.

Table 6- 4 Security levels for the CPU

Security level	Access restrictions
No protection	Allows full access without password-protection.
Write protection	Allows HMI access and all forms of PLC-to-PLC communications without password-protection. Password is required for modifying (writing to) the CPU and for changing the CPU mode (RUN/STOP).
Read/write protection	Allows HMI access and all forms of PLC-to-PLC communications without password-protection. Password is required for reading the data in the CPU, for modifying (writing to) the CPU, and for changing the CPU mode (RUN/STOP).

6.6.2 Know-how protection

Know-how protection allows you to prevent one or more code blocks (OB, FB, FC, or DB) in your program from unauthorized access. You create a password to limit access to the code block. The password-protection prevents unauthorized reading or modification of the code block. Without the password, you can read only the following information about the code block:

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

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

Use the "Properties" task card of the code block to configure the know-how protection for that block. After opening the code block, select "Protection" from Properties.

1. In the Properties for the code block, click the "Protection" button to display the "Know-how protection" dialog.
2. Click the "Define" button to enter the password.

After entering and confirming the password, click "OK".

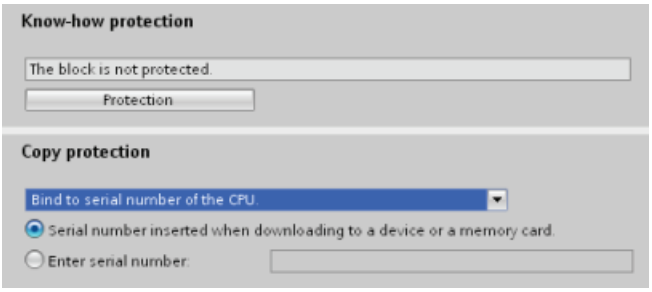
6.6.3 Copy protection

An additional security feature allows you to bind the program or code blocks for use with a specific memory card or CPU. This feature is especially useful for protecting your intellectual property. When you bind a program or block to a specific device, you restrict the program or code block for use only with a specific memory card or CPU. This feature allows you to distribute a program or code block electronically (such as over the Internet or through email) or by sending a memory cartridge.

Use the "Properties" task card of the code block to bind the block to a specific CPU or memory card.

1. After opening the code block, select "Protection".

2. From the drop-down list under "Copy protection" task, select the option to bind the code block either to a memory card or to a specific CPU.



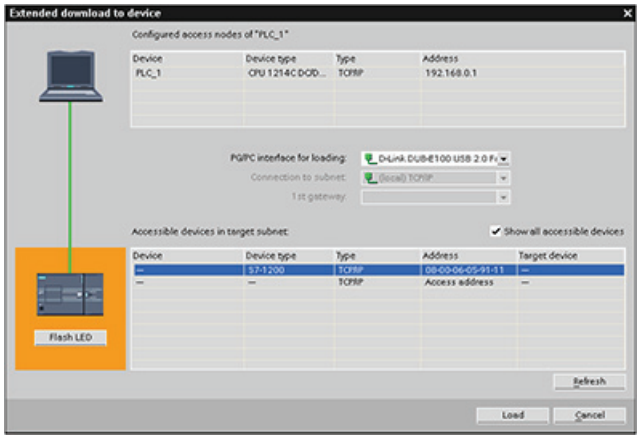
3. Select the type of copy protection and enter the serial number for the memory card or CPU.

Note

The serial number is case-sensitive.

6.7 Downloading the elements of your program

You can download the elements of your project from the programming device to the CPU. When you download a project, the CPU stores the user program (OBs, FCs, FBs and DBs) in permanent memory.



You can download your project from the programming device to your CPU from any of the following locations:

- "Project tree": Right-click the program element, and then click the context-sensitive "Download" selection.
- "Online" menu: Click the "Download to device" selection.
- Toolbar: Click the "Download to device" icon.

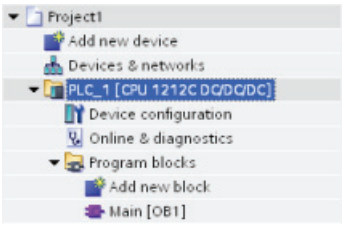
6.8 Uploading from the CPU

6.8.1 Copying elements of the project

You can also copy the program blocks from an online CPU or a memory card attached to your programming device.

Prepare the offline project for the copied program blocks:

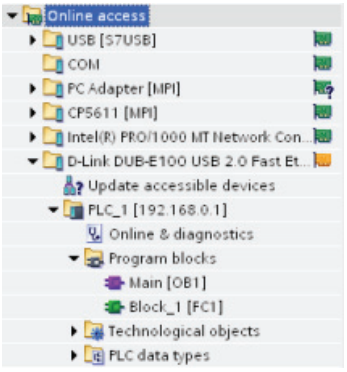
1. Add a CPU device that matches the online CPU.
2. Expand the CPU node once so that the "Program blocks" folder is visible.



To upload the program blocks from the online CPU to the offline project, follow these steps:

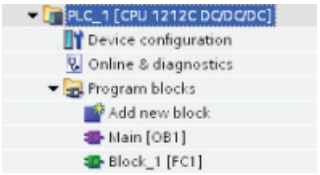
1. Click the "Program blocks" folder in the offline project.
2. Click the "Go online" button.
3. Click the "Upload" button.
4. Confirm your decision from the Upload dialog (Page 675).

As an alternative to the previous method, follow these steps:



1. From the project navigator, expand the node for "Online access" to select the program blocks in the online CPU:
2. Expand the node for the network, and double click "Update accessible devices".
3. Expand the node for the CPU.
4. Drag the "Program blocks" folder from the online CPU and drop the folder into the "Program blocks" folder of your offline project.
5. In the "Upload preview" dialog, select the box for "Continue", and then click the "Upload from device" button.

When the upload is complete, all of the program blocks, technology blocks, and tags will be displayed in the offline area.



Note

You can copy the program blocks from the online CPU to an existing program. The "Program-blocks" folder of the offline project does not have to be empty. However, the existing program will be deleted and replaced by the user program from the online CPU.

6.8.2 Using the compare function

You can use the "Compare" editor (Page 681) in STEP 7 to find differences between the online and offline projects. You might find this useful prior to uploading from the CPU.

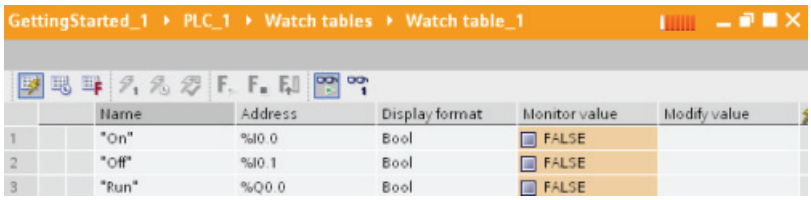
6.9 Debugging and testing the program

6.9.1 Monitor and modify data in the CPU

As shown in the following table, you can monitor and modify values in the online CPU.

Table 6- 5 Monitoring and modifying data with STEP 7

Editor	Monitor	Modify	Force
Watch table	Yes	Yes	No
Force table	Yes	No	Yes
Program editor	Yes	Yes	No
Tag table	Yes	No	No
DB editor	Yes	No	No



Monitoring with a watch table

Monitoring with the LAD editor

Refer to the "Online and diagnostics" chapter for more information about monitoring and modifying data in the CPU (Page 682).

6.9.2 Watch tables and force tables

You use "watch tables" for monitoring and modifying the values of a user program being executed by the online CPU. You can create and save different watch tables in your project to support a variety of test environments. This allows you to reproduce tests during commissioning or for service and maintenance purposes.

With a watch table, you can monitor and interact with the CPU as it executes the user program. You can display or change values not only for the tags of the code blocks and data blocks, but also for the memory areas of the CPU, including the inputs and outputs (I and Q), peripheral inputs (I:P), bit memory (M), and data blocks (DB).

With the watch table, you can enable the physical outputs (Q:P) of a CPU in STOP mode. For example, you can assign specific values to the outputs when testing the wiring for the CPU.

STEP 7 also provides a force table for "forcing" a tag to a specific value. For more information about forcing, see the section on forcing values in the CPU (Page 689) in the "Online and Diagnostics" chapter.

Note

The force values are stored in the CPU and not in the watch table.

You cannot force an input (or "I" address). However, you can force a peripheral input. To force a peripheral input, append a ":P" to the address (for example: "On:P").

6.9.3 Cross reference to show usage

The Inspector window displays cross-reference information about how a selected object is used throughout the complete project, such as the user program, the CPU and any HMI devices. The "Cross-reference" tab displays the instances where a selected object is being used and the other objects using it. The Inspector window also includes blocks which are only available online in the cross-references. To display the cross-references, select the "Show cross-references" command. (In the Project view, find the cross references in the "Tools" menu.)

Note

You do not have to close the editor to see the cross-reference information.

You can sort the entries in the cross-reference. The cross-reference list provides an overview of the use of memory addresses and tags within the user program.

- When creating and changing a program, you retain an overview of the operands, tags and block calls you have used.
- From the cross-references, you can jump directly to the point of use of operands and tags.
- During a program test or when troubleshooting, you are notified about which memory location is being processed by which command in which block, which tag is being used in which screen, and which block is called by which other block.

Table 6- 6 Elements of the cross reference

Column	Description
Object	Name of the object that uses the lower-level objects or that is being used by the lower-level objects
Quantity	Number of uses
Location	Each location of use, for example, network
Property	Special properties of referenced objects, for example, the tag names in multi-instance declarations
as	Shows additional information about the object, such as whether an instance DB is used as template or as a multiple instance
Access	Type of access, whether access to the operand is read access (R) and/or write access (W)
Address	Address of the operand
Type	Information on the type and language used to create the object
Path	Path of object in project tree

6.9.4 Call structure to examine the calling hierarchy

The call structure describes the call hierarchy of the block within your user program. It provides an overview of the blocks used, calls to other blocks, the relationships between blocks, the data requirements for each block, and the status of the blocks. You can open the program editor and edit blocks from the call structure.

Displaying the call structure provides you with a list of the blocks used in the user program. STEP 7 highlights the first level of the call structure and displays any blocks that are not called by any other block in the program. The first level of the call structure displays the OBs and any FCs, FBs, and DBs that are not called by an OB. If a code block calls another block, the called block is shown as an indentation under the calling block. The call structure only displays those blocks that are called by a code block.

You can selectively display only the blocks causing conflicts within the call structure. The following conditions cause conflicts:

- Blocks that execute any calls with older or newer code time stamps
- Blocks that call a block with modified interface
- Blocks that use a tag with modified address and/or data type
- Blocks that are called neither directly nor indirectly by an OB
- Blocks that call a non-existent or missing block

You can group several block calls and data blocks as a group. You use a drop-down list to see the links to the various call locations.

You can also perform a consistency check to show time stamp conflicts. Changing the time stamp of a block during or after the program is generated can lead to time stamp conflicts, which in turn cause inconsistencies among the blocks that are calling and being called.

- Most time stamp and interface conflicts can be corrected by recompiling the code blocks.
- If compilation fails to clear up inconsistencies, use the link in the "Details" column to go to the source of the problem in the program editor. You can then manually eliminate any inconsistencies.
- Any blocks marked in red must be recompiled.

Basic instructions

7

7.1 Bit logic

7.1.1 Bit logic contacts and coils

LAD and FBD are very effective for handling Boolean logic. While SCL is especially effective for complex mathematical computation and for project control structures, you can use SCL for Boolean logic.

LAD contacts

Table 7- 1 Normally open and normally closed contacts

LAD	SCL	Description

	`IF in THEN`
	`  Statement;`
	`ELSE`
	`  Statement;`
	`END_IF;`
		Normally open and normally closed contacts: You can connect contacts to other contacts and create your own combination logic. If the input bit you specify uses memory identifier I (input) or Q (output), then the bit value is read from the process-image register. The physical contact signals in your control process are wired to I terminals on the PLC. The CPU scans the wired input signals and continuously updates the corresponding state values in the process-image input register.
		You can specify an immediate read of a physical input using ":P" following the I offset (example: "%I3.4:P"). For an immediate read, the bit data values are read directly from the physical input instead of the process image. An immediate read does not update the process image.

	`IF NOT (in) THEN`
	`  Statement;`
	`ELSE`
	`  Statement;`
	`END_IF;`	

Table 7- 2 Data types for the parameters

Parameter	Data type	Description
IN	Bool	Assigned bit

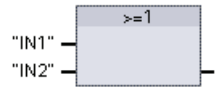
- The Normally Open contact is closed (ON) when the assigned bit value is equal to 1.
- The Normally Closed contact is closed (ON) when the assigned bit value is equal to 0.
- Contacts connected in series create AND logic networks.
- Contacts connected in parallel create OR logic networks.

FBD AND, OR, and XOR boxes

In FBD programming, LAD contact networks are transformed into AND (&), OR (>=1), and exclusive OR (x) box networks where you can specify bit values for the box inputs and outputs. You may also connect to other logic boxes and create your own logic combinations. After the box is placed in your network, you can drag the "Insert input" tool from the "Favorites" toolbar or instruction tree and then drop it onto the input side of the box to add more inputs. You can also right-click on the box input connector and select "Insert input".

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

Table 7- 3 AND, OR, and XOR boxes

FBD	SCL[1]	Description
	`out := in1 AND in2;`	All inputs of an AND box must be TRUE for the output to be TRUE.
	`out := in1 OR in2;`	Any input of an OR box must be TRUE for the output to be TRUE.
	`out := in1 XOR in2;`	An odd number of the inputs of an XOR box must be TRUE for the output to be TRUE.

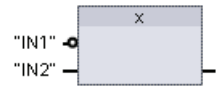
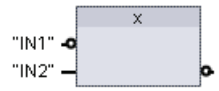
[1] For SCL: You must assign the result of the operation to a variable to be used for another statement.

Table 7- 4 Data types for the parameters

Parameter	Data type	Description
IN1, IN2	Bool	Input bit

NOT logic inverter

Table 7- 5 NOT Logic inverter

LAD	FBD	SCL	Description
	 	**NOT**	For FBD programming, you can drag the "Negate binary input" tool from the "Favorites" toolbar or instruction tree and then drop it on an input or output to create a logic inverter on that box connector. The LAD NOT contact inverts the logical state of power flow input. • If there is no power flow into the NOT contact, then there is power flow out. • If there is power flow into the NOT contact, then there is no power flow out.

Output coil and assignment box

The coil output instruction writes a value for an output bit. If the output bit you specify uses memory identifier Q, then the CPU turns the output bit in the process-image register on or off, setting the specified bit equal to power flow status. The output signals for your control actuators are wired to the Q terminals of the CPU. In RUN mode, the CPU system continuously scans your input signals, processes the input states according to your program logic, and then reacts by setting new output state values in the process-image output register. After each program execution cycle, the CPU system transfers the new output state reaction stored in the process-image register to the wired output terminals.

Table 7- 6 Output coil (LAD) and output assignment box (FBD)

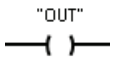
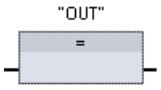
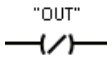
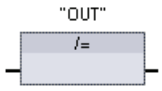
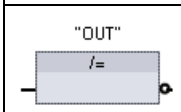
LAD	FBD	SCL	Description
		**out := <Boolean expression>;**	In FBD programming, LAD coils are transformed into assignment (= and /=) boxes where you specify a bit address for the box output. Box inputs and outputs can be connected to other box logic or you can enter a bit address. You can specify an immediate write of a physical output using ":P" following the Q offset (example: "%Q3.4:P"). For an immediate write, the bit data values are written to the process image output and directly to physical output.
	 	**out := NOT <Boolean expression>;**	

Table 7- 7 Data types for the parameters

Parameter	Data type	Description
OUT	Bool	Assigned bit

- If there is power flow through an output coil or an FBD "=" box is enabled, then the output bit is set to 1.
- If there is no power flow through an output coil or an FBD "=" assignment box is not enabled, then the output bit is set to 0.
- If there is power flow through an inverted output coil or an FBD "/=" box is enabled, then the output bit is set to 0.
- If there is no power flow through an inverted output coil or an FBD "/=" box is not enabled, then the output bit is set to 1.

7.1.2 Set and reset instructions

Set and Reset 1 bit

Table 7- 8 S and R instructions

LAD	FBD	SCL	Description
"OUT" —(S)—	"OUT" S "IN"	Not available	When S (Set) is activated, then the data value at the OUT address is set to 1. When S is not activated, OUT is not changed.
"OUT" —(R)—	"OUT" R "IN"	Not available	When R (Reset) is activated, then the data value at the OUT address is set to 0. When R is not activated, OUT is not changed.

[1] For LAD and FBD: These instructions can be placed anywhere in the network.

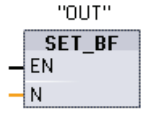
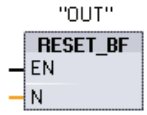
[2] For SCL: You must write code to replicate this function within your application.

Table 7- 9 Data types for the parameters

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

Set and Reset Bit Field

Table 7- 10 SET_BF and RESET_BF instructions

LAD[1]	FBD	SCL	Description
		Not available	When SET_BF is activated, a data value of 1 is assigned to "n" bits starting at address OUT. When SET_BF is not activated, OUT is not changed.
		Not available	RESET_BF writes a data value of 0 to "n" bits starting at address OUT. When RESET_BF is not activated, OUT is not changed.

1 For LAD and FBD: These instructions must be the right-most instruction in a branch.

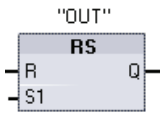
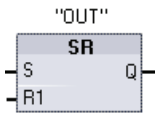
2 For SCL: You must write code to replicate this function within your application.

Table 7- 11 Data types for the parameters

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

Set-dominant and Reset-dominant bit latches

Table 7- 12 RS and SR instructions

LAD / FBD	SCL	Description
	Not available	RS is a set dominant latch where the set dominates. If the set (S1) and reset (R) signals are both true, the output address OUT will be 1.
	Not available	SR is a reset dominant latch where the reset dominates. If the set (S) and reset (R1) signals are both true, the output address OUT will be 0.

1 For LAD and FBD: These instructions must be the right-most instruction in a branch.

2 For SCL: You must write code to replicate this function within your application.

Table 7- 13 Data types for the parameters

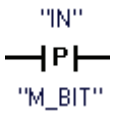
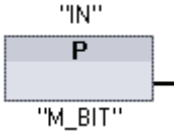
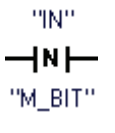
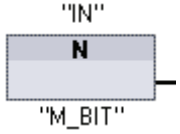
Parameter	Data type	Description
S, S1	Bool	Set input; 1 indicates dominance
R, R1	Bool	Reset input; 1 indicates dominance
OUT	Bool	Assigned bit output "OUT"
Q	Bool	Follows state of "OUT" bit

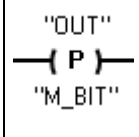
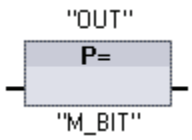
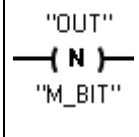
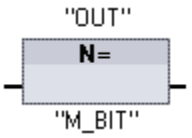
The OUT parameter specifies the bit address that is set or reset. The optional OUT output Q reflects the signal state of the "OUT" address.

Instruction	S1	R	"OUT" bit
RS	0	0	Previous state
	0	1	0
	1	0	1
	1	1	1
	S	**R1**	
SR	0	0	Previous state
	0	1	0
	1	0	1
	1	1	0

7.1.3 Positive and negative edge instructions

Table 7- 14 Positive and negative transition detection

LAD	FBD	SCL	Description
		Not available	LAD: The state of this contact is TRUE when a positive transition (OFF-to-ON) is detected on the assigned "IN" bit. The contact logic state is then combined with the power flow in state to set the power flow out state. The P contact can be located anywhere in the network except the end of a branch. FBD: The output logic state is TRUE when a positive transition (OFF-to-ON) is detected on the assigned input bit. The P box can only be located at the beginning of a branch.
		Not available	LAD: The state of this contact is TRUE when a negative transition (ON-to-OFF) is detected on the assigned input bit. The contact logic state is then combined with the power flow in state to set the power flow out state. The N contact can be located anywhere in the network except the end of a branch. FBD: The output logic state is TRUE when a negative transition (ON-to-OFF) is detected on the assigned input bit. The N box can only be located at the beginning of a branch.

LAD	FBD	SCL	Description
		Not available	LAD: The assigned bit "OUT" is TRUE when a positive transition (OFF-to-ON) is detected on the power flow entering the coil. The power flow in state always passes through the coil as the power flow out state. The P coil can be located anywhere in the network. FBD: The assigned bit "OUT" is TRUE when a positive transition (OFF-to-ON) is detected on the logic state at the box input connection or on the input bit assignment if the box is located at the start of a branch. The input logic state always passes through the box as the output logic state. The P= box can be located anywhere in the branch.
		Not available	LAD: The assigned bit "OUT" is TRUE when a negative transition (ON-to-OFF) is detected on the power flow entering the coil. The power flow in state always passes through the coil as the power flow out state. The N coil can be located anywhere in the network. FBD: The assigned bit "OUT" is TRUE when a negative transition (ON-to-OFF) is detected on the logic state at the box input connection or on the input bit assignment if the box is located at the start of a branch. The input logic state always passes through the box as the output logic state. The N= box can be located anywhere in the branch.

[1] For SCL: You must write code to replicate this function within your application.

Table 7- 15 P_TRIG and N_TRIG instructions

LAD / FBD	SCL	Description
	Not available	The Q output power flow or logic state is TRUE when a positive transition (OFF-to-ON) is detected on the CLK input state (FBD) or CLK power flow in (LAD). In LAD, the P_TRIG instruction cannot be located at the beginning or end of a network. In FBD, the P_TRIG instruction can be located anywhere except the end of a branch.
	Not available	The Q output power flow or logic state is TRUE when a negative transition (ON-to-OFF) is detected on the CLK input state (FBD) or CLK power flow in (LAD). In LAD, the N_TRIG instruction cannot be located at the beginning or end of a network. In FBD, the N_TRIG instruction can be located anywhere except the end of a branch.

[1] For SCL: You must write code to replicate this function within your application.

Table 7- 16 Data types for the parameters (P and N contacts/coils, P=, N=, P_TRIG and N_TRIG)

Parameter	Data type	Description
M_BIT	Bool	Memory bit in which the previous state of the input is saved
IN	Bool	Input bit whose transition edge is to be detected
OUT	Bool	Output bit which indicates a transition edge was detected
CLK	Bool	Power flow or input bit whose transition edge is to be detected
Q	Bool	Output which indicates an edge was detected

All edge instructions use a memory bit (M_BIT) to store the previous state of the input signal being monitored. An edge is detected by comparing the state of the input with the state of the memory bit. If the states indicate a change of the input in the direction of interest, then an edge is reported by writing the output TRUE. Otherwise, the output is written FALSE.

Note

Edge instructions evaluate the input and memory-bit values each time they are executed, including the first execution. You must account for the initial states of the input and memory bit in your program design either to allow or to avoid edge detection on the first scan.

Because the memory bit must be maintained from one execution to the next, you should use a unique bit for each edge instruction, and you should not use this bit any other place in your program. You should also avoid temporary memory and memory that can be affected by other system functions, such as an I/O update. Use only M, global DB, or Static memory (in an instance DB) for M_BIT memory assignments.

7.2 Timers

You use the timer instructions to create programmed time delays. The number of timers that you can use in your user program is limited only by the amount of memory in the CPU. Each timer uses a 16 byte IEC_Timer data type DB structure to store timer data that is specified at the top of the box or coil instruction. STEP 7 automatically creates the DB when you insert the instruction.

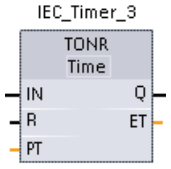
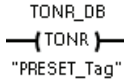
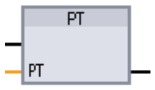
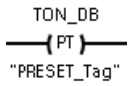
Table 7- 17 Timer instructions

LAD / FBD boxes	LAD coils	SCL	Description

IEC_Timer_0
TP
Time
IN Q
PT ET
	TP_DB
	TP
	"PRESET_Tag"
		`"IEC_Timer_0_DB".TP(`
		`IN:=_bool_in_,`
		`PT:=_time_in_,`
		`Q=>_bool_out_,`
		`ET=>_time_out_);`
			The TP timer generates a pulse with a preset width time.

IEC_Timer_1
TON
Time
IN Q
PT ET
	TON_DB
	TON
	"PRESET_Tag"
		`"IEC_Timer_0_DB".TON (`
		`IN:=_bool_in_,`
		`PT:=_time_in_,`
		`Q=>_bool_out_,`
		`ET=>_time_out_);`
			The TON timer sets output Q to ON after a preset time delay.

IEC_Timer_2
TOF
Time
IN Q
PT ET
	TOF_DB
	TOF
	"PRESET_Tag"
		`"IEC_Timer_0_DB".TOF (`
		`IN:=_bool_in_,`
		`PT:=_time_in_,`
		`Q=>_bool_out_,`
		`ET=>_time_out_);`
			The TOF timer resets output Q to OFF after a preset time delay.

LAD / FBD boxes	LAD coils	SCL	Description
		`"IEC_Timer_0_DB".TONR ( IN:=_bool_in_, R:=_bool_in_ PT:=_time_in_, Q=>_bool_out_, ET=>_time_out_);`	The TONR timer sets output Q to ON after a preset time delay. Elapsed time is accumulated over multiple timing periods until the R input is used to reset the elapsed time.
FBD only: 		(No SCL equivalent)	The PT (Preset timer) coil loads a new PRESET time value in the specified IEC_Timer.
FBD only: 		(No SCL equivalent)	The RT (Reset timer) coil resets the specified IEC_Timer.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL examples, "IEC_Timer_0_DB" is the name of the instance DB.

Table 7- 18 Data types for the parameters

Parameter	Data type	Description
Box: IN Coil: Power flow	Bool	TP, TON, and TONR: Box: 0=Disable timer, 1=Enable timer Coil: No power flow=Disable timer, Power flow=Enable timer TOF: Box: 0=Enable timer, 1=Disable timer Coil: No power flow=Enable timer, Power flow=Disable timer
R	Bool	TONR box only: 0=No reset 1= Reset elapsed time and Q bit to 0
Box: PT Coil: "PRESET_Tag"	Time	Timer box or coil: Preset time input
Box: Q Coil: DBdata.Q	Bool	Timer box: Q box output or Q bit in the timer DB data Timer coil: you can only address the Q bit in the timer DB data
Box: ET Coil: DBdata.ET	Time	Timer box: ET (elapsed time) box output or ET time value in the timer DB data Timer coil: you can only address the ET time value in the timer DB data.

Table 7- 19 Effect of value changes in the PT and IN parameters

Timer	Changes in the PT and IN box parameters and the corresponding coil parameters
TP	• Changing PT has no effect while the timer runs. • Changing IN has no effect while the timer runs.
TON	• Changing PT has no effect while the timer runs. • Changing IN to FALSE, while the timer runs, resets and stops the timer.
TOF	• Changing PT has no effect while the timer runs. • Changing IN to TRUE, while the timer runs, resets and stops the timer.
TONR	• Changing PT has no effect while the timer runs, but has an effect when the timer resumes. • Changing IN to FALSE, while the timer runs, stops the timer but does not reset the timer. Changing IN back to TRUE will cause the timer to start timing from the accumulated time value.

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

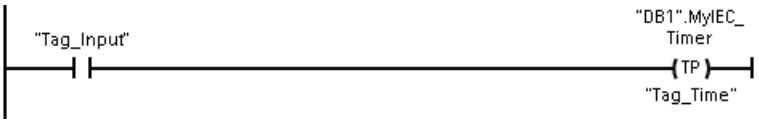
Table 7- 20 Size and range of the TIME data type

Data type	Size	Valid number ranges[1]
TIME	32 bits, stored as DInt data	T#-24d_20h_31m_23s_648ms to T#24d_20h_31m_23s_647ms Stored as -2,147,483,648 ms to +2,147,483,647 ms

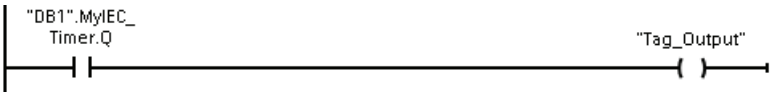
[1] The negative range of the TIME data type shown above cannot be used with the timer instructions. Negative PT (preset time) values are set to zero when the timer instruction is executed. ET (elapsed time) is always a positive value.

Timer coil example

The -(TP)-, -(TON)-, -(TOF)-, and -(TONR)- timer coils must be the last instruction in a LAD network. As shown in the timer example, a contact instruction in a subsequent network evaluates the Q bit in a timer coil's IEC_Timer DB data. Likewise, you must address the ELAPSED element in the IEC_timer DB data if you want to use the elapsed time value in your program.



The pulse timer is started on a 0 to 1 transition of the Tag_Input bit value. The timer runs for the time specified by Tag_Time time value.



As long as the timer runs, the state of DB1.MyIEC_Timer.Q=1 and the Tag_Output value=1. When the Tag_Time value has elapsed, then DB1.MyIEC_Timer.Q=0 and the Tag_Output value=0.

Reset timer -(RT)- and Preset timer -(PT)- coils

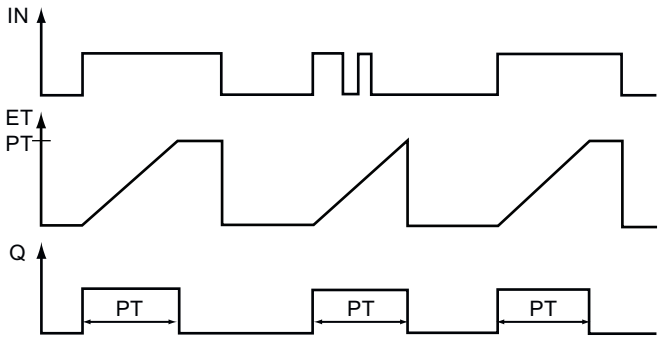
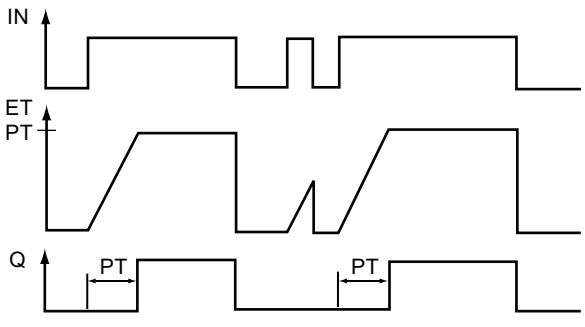
These coil instructions can be used with box or coil timers and can be placed in a mid-line position. The coil output power flow status is always the same as the coil input status. When the -(RT)- coil is activated, the ELAPSED time element of the specified IEC_Timer DB data is reset to 0. When the -(PT)- coil is activated, the PRESET time element of the specified IEC_Timer DB data is reset to 0.

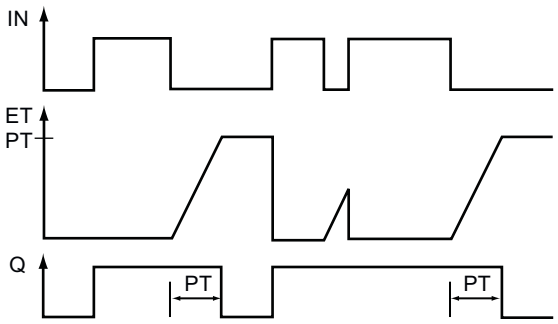
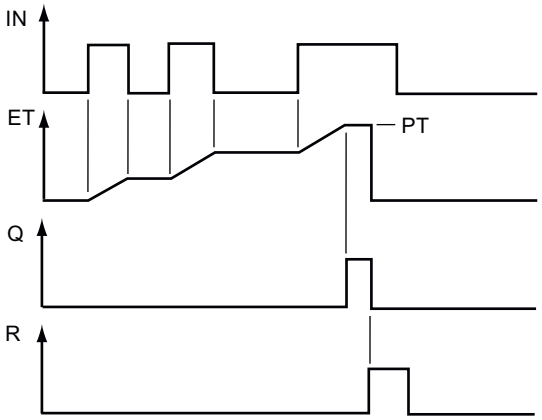
Note

When you place timer instructions in an FB, you can select the "Multi-instance data block" option. The timer structure names can be different with separate data structures, but the timer data is contained in a single data block and does not require a separate data block for each timer. This reduces the processing time and data storage necessary for handling the timers. There is no interaction between the timer data structures in the shared multi-instance DB.

Operation of the timers

Table 7- 21 Types of IEC timers

Timer	Timing diagram
TP: Pulse timer The TP timer generates a pulse with a preset width time.	
TON: ON-delay timer The TON timer sets output Q to ON after a preset time delay.	

Timer	Timing diagram
TOF: OFF-delay timer The TOF timer resets output Q to OFF after a preset time delay.	
TONR: ON-delay Retentive timer The TONR timer sets output Q to ON after a preset time delay. Elapsed time is accumulated over multiple timing periods until the R input is used to reset the elapsed time.	

Note

In the CPU, no dedicated resource is allocated to any specific timer instruction. Instead, each timer utilizes its own timer structure in DB memory and a continuously-running internal CPU timer to perform timing.

When a timer is started due to an edge change on the input of a TP, TON, TOF, or TONR instruction, the value of the continuously-running internal CPU timer is copied into the START member of the DB structure allocated for this timer instruction. This start value remains unchanged while the timer continues to run, and is used later each time the timer is updated. Each time the timer is started, a new start value is loaded into the timer structure from the internal CPU timer.

When a timer is updated, the start value described above is subtracted from the current value of the internal CPU timer to determine the elapsed time. The elapsed time is then compared with the preset to determine the state of the timer Q bit. The ELAPSED and Q members are then updated in the DB structure allocated for this timer. Note that the elapsed time is clamped at the preset value (the timer does not continue to accumulate elapsed time after the preset is reached).

A timer update is performed when and only when:

- A timer instruction (TP, TON, TOF, or TONR) is executed
- The "ELAPSED" member of the timer structure in DB is referenced directly by an instruction
- The "Q" member of the timer structure in DB is referenced directly by an instruction

Timer programming

The following consequences of timer operation should be considered when planning and creating your user program:

- You can have multiple updates of a timer in the same scan. The timer is updated each time the timer instruction (TP, TON, TOF, TONR) is executed and each time the ELAPSED or Q member of the timer structure is used as a parameter of another executed instruction. This is an advantage if you want the latest time data (essentially an immediate read of the timer). However, if you desire to have consistent values throughout a program scan, then place your timer instruction prior to all other instructions that need these values, and use tags from the Q and ET outputs of the timer instruction instead of the ELAPSED and Q members of the timer DB structure.
- You can have scans during which no update of a timer occurs. It is possible to start your timer in a function, and then cease to call that function again for one or more scans. If no other instructions are executed which reference the ELAPSED or Q members of the timer structure, then the timer will not be updated. A new update will not occur until either the timer instruction is executed again or some other instruction is executed using ELAPSED or Q from the timer structure as a parameter.
- Although not typical, you can assign the same DB timer structure to multiple timer instructions. In general, to avoid unexpected interaction, you should only use one timer instruction (TP, TON, TOF, TONR) per DB timer structure.
- Self-resetting timers are useful to trigger actions that need to occur periodically. Typically, self-resetting timers are created by placing a normally-closed contact which references the timer bit in front of the timer instruction. This timer network is typically located above one or more dependent networks that use the timer bit to trigger actions. When the timer expires (elapsed time reaches preset value), the timer bit is ON for one scan, allowing the dependent network logic controlled by the timer bit to execute. Upon the next execution of the timer network, the normally closed contact is OFF, thus resetting the timer and clearing the timer bit. The next scan, the normally closed contact is ON, thus restarting the timer. When creating self-resetting timers such as this, do not use the "Q" member of the timer DB structure as the parameter for the normally-closed contact in front of the timer instruction. Instead, use the tag connected to the "Q" output of the timer instruction for this purpose. The reason to avoid accessing the Q member of the timer DB structure is because this causes an update to the timer and if the timer is updated due to the normally closed contact, then the contact will reset the timer instruction immediately. The Q output of the timer instruction will not be ON for the one scan and the dependent networks will not execute.

Time data retention after a RUN-STOP-RUN transition or a CPU power cycle

If a run mode session is ended with stop mode or a CPU power cycle and a new run mode session is started, then the timer data stored in the previous run mode session is lost, unless the timer data structure is specified as retentive (TP, TON, TOF, and TONR timers).

When you accept the defaults in the call options dialog after you place a timer instruction in the program editor, you are automatically assigned an instance DB which **cannot be made retentive**. To make your timer data retentive, you must either use a global DB or a Multi-instance DB.

Assign a global DB to store timer data as retentive data

This option works regardless of where the timer is placed (OB, FC, or FB).

1. Create a global DB:
   - Double-click "Add new block" from the Project tree
   - Click the data block (DB) icon
   - For the Type, choose global DB
   - If you want to be able to select individual data elements in this DB as retentive, be sure the DB type "Optimized" box is checked. The other DB type option "Standard - compatible with S7-300/400" only allows setting all DB data elements retentive or none retentive.
   - Click OK
2. Add timer structure(s) to the DB:
   - In the new global DB, add a new static tag using data type IEC_Timer.
   - In the "Retain" column, check the box so that this structure will be retentive.
   - Repeat this process to create structures for all the timers that you want to store in this DB. You can either place each timer structure in a unique global DB, or you can place multiple timer structures into the same global DB. You can also place other static tags besides timers in this global DB. Placing multiple timer structures into the same global DB allows you to reduce your overall number of blocks.
   - Rename the timer structures if desired.
3. Open the program block for editing where you want to place a retentive timer (OB, FC, or FB).
4. Place the timer instruction at the desired location.
5. When the call options dialog appears, click the cancel button.
6. On the top of the new timer instruction, type the name (do not use the helper to browse) of the global DB and timer structure that you created above (example: "Data_block_3.Static_1").

Assign a multi-instance DB to store timer data as retentive data

This option only works if you place the timer in an FB.

This option depends upon whether the FB was created with "Optimized" block access (allows symbolic access only). Once the FB has been created, you cannot change the checkbox for "Optimized"; it must be chosen correctly when the FB is created, on the first screen after selecting "Add new block" from the tree. To verify how the access attribute is configured for an existing FB, right-click on the FB in the Project tree, choose properties, and then choose attributes.

If the FB was created with the "Optimized" box checked (allows symbolic access only):

1. Open the FB for edit.
2. Place the timer instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the Multi instance icon. The Multi Instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the timer if desired.
5. Click OK. The timer instruction appears in the editor, and the IEC_TIMER structure appears in the FB Interface under Static.
6. If necessary, open the FB interface editor (may have to click on the small arrow to expand the view).
7. Under Static, locate the timer structure that was just created for you.
8. In the Retain column for this timer structure, change the selection to "Retain". Whenever this FB is called later from another program block, an instance DB will be created with this interface definition which contains the timer structure marked as retentive.

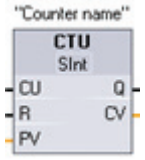
If the FB was created with the "Standard - compatible with S7-300/400" box checked (allows symbolic and direct access):

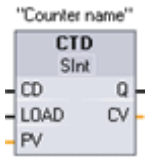
1. Open the FB for edit.
2. Place the timer instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the multi instance icon. The multi instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the timer if desired.
5. Click OK. The timer instruction appears in the editor, and the IEC_TIMER structure appears in the FB Interface under Static.
6. Open the block that will use this FB.
7. Place this FB at the desired location. Doing so results in the creation of an instance data block for this FB.
8. Open the instance data block created when you placed the FB in the editor.
9. Under Static, locate the timer structure of interest. In the Retain column for this timer structure, check the box to make this structure retentive.

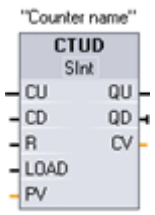
7.3 Counters

Table 7- 22 Counter instructions

LAD / FBD	SCL	Description

	`"IEC_Counter_0_DB".CTU(`
	`CU:=_bool_in,`
	`R:=_bool_in,`
	`PV:=_int_in,`
	`Q=>_bool_out,`
	`CV=>_int_out);`
		Use the counter instructions to count internal program events and external process events. Each counter uses a structure stored in a data block to maintain counter data. You assign the data block when the counter instruction is placed in the editor.
		• CTU is a count-up counter
		• CTD is a count-down counter
		• CTUD is a count-up-and-down counter

	`"IEC_Counter_0_DB".CTD(`
	`CD:=_bool_in,`
	`LD:=_bool_in,`
	`PV:=_int_in,`
	`Q=>_bool_out,`
	`CV=>_int_out);`	

	`"IEC_Counter_0_DB".CTUD(`
	`CU:=_bool_in,`
	`CD:=_bool_in,`
	`R:=_bool_in,`
	`LD:=_bool_in,`
	`PV:=_int_in,`
	`QU=>_bool_out,`
	`QD=>_bool_out,`
	`CV=>_int_out);`	

1 For LAD and FBD: Select the count value data type from the drop-down list below the instruction name.

2 STEP 7 automatically creates the DB when you insert the instruction.

3 In the SCL examples, "IEC_Counter_0_DB" is the name of the instance DB.

Table 7- 23 Data types for the parameters

Parameter	Data type[1]	Description
CU, CD	Bool	Count up or count down, by one count
R (CTU, CTUD)	Bool	Reset count value to zero
LD (CTD, CTUD)	Bool	Load control for preset value
PV	SInt, Int, DInt, USInt, UInt, UDInt	Preset count value
Q, QU	Bool	True if CV >= PV
QD	Bool	True if CV <= 0
CV	SInt, Int, DInt, USInt, UInt, UDInt	Current count value

1 The numerical range of count values depends on the data type you select. If the count value is an unsigned integer type, you can count down to zero or count up to the range limit. If the count value is a signed integer, you can count down to the negative integer limit and count up to the positive integer limit.

The number of counters that you can use in your user program is limited only by the amount of memory in the CPU. Counters use the following amount of memory:

- For SInt or USInt data types, the counter instruction uses 3 bytes.
- For Int or UInt data types, the counter instruction uses 6 bytes.
- For DInt or UDInt data types, the counter instruction uses 12 bytes.

These instructions use software counters whose maximum counting rate is limited by the execution rate of the OB in which they are placed. The OB that the instructions are placed in must be executed often enough to detect all transitions of the CU or CD inputs. For faster counting operations, see the CTRL_HSC instruction (Page 337).

Note

When you place counter instructions in an FB, you can select the multi-instance DB option, the counter structure names can be different with separate data structures, but the counter data is contained in a single DB and does not require a separate DB for each counter. This reduces the processing time and data storage necessary for the counters. There is no interaction between the counter data structures in the shared multi-instance DB.

Operation of the counters

Table 7- 24 Operation of the CTU counter

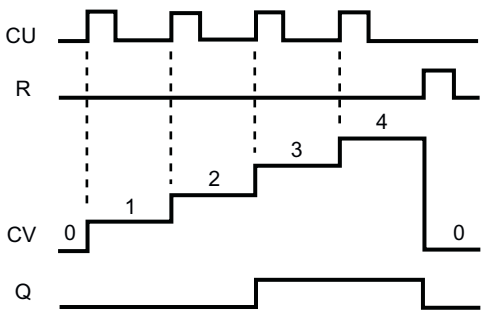
Counter	Operation
The CTU counter counts up by 1 when the value of parameter CU changes from 0 to 1. The CTU timing diagram shows the operation for an unsigned integer count value (where PV = 3). • If the value of parameter CV (current count value) is greater than or equal to the value of parameter PV (preset count value), then the counter output parameter Q = 1. • If the value of the reset parameter R changes from 0 to 1, then the current count value is reset to 0.	

Table 7- 25 Operation of the CTD counter

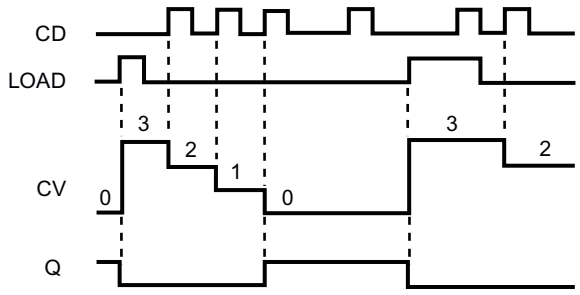
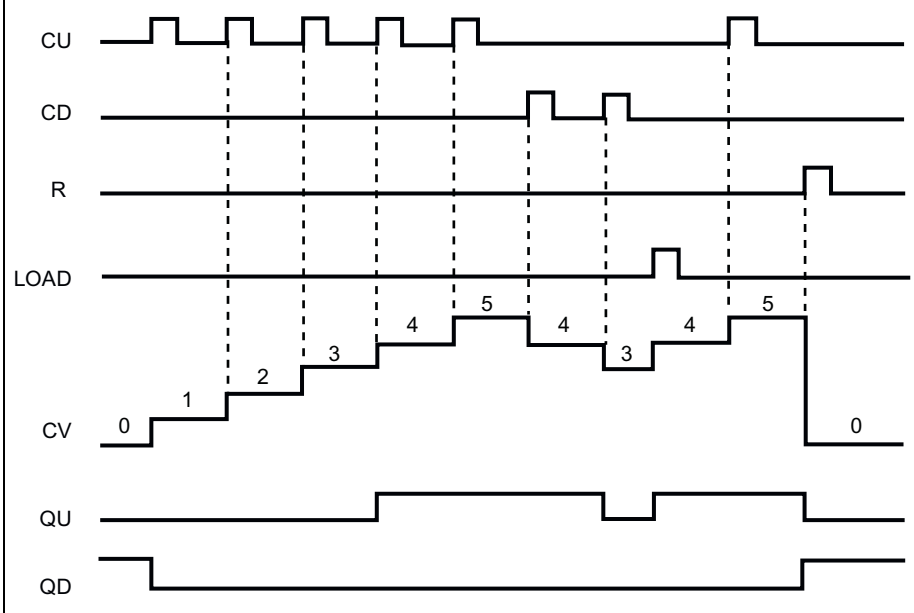
Counter	Operation
The CTD counter counts down by 1 when the value of parameter CD changes from 0 to 1. The CTD timing diagram shows the operation for an unsigned integer count value (where PV = 3). • If the value of parameter CV (current count value) is equal to or less than 0, the counter output parameter Q = 1. • If the value of parameter LOAD changes from 0 to 1, the value at parameter PV (preset value) is loaded to the counter as the new CV (current count value).	

Table 7- 26 Operation of the CTUD counter

Counter	Operation
The CTUD counter counts up or down by 1 on the 0 to 1 transition of the count up (CU) or count down (CD) inputs. The CTUD timing diagram shows the operation for an unsigned integer count value (where PV = 4). • If the value of parameter CV is equal to or greater than the value of parameter PV, then the counter output parameter QU = 1. • If the value of parameter CV is less than or equal to zero, then the counter output parameter QD = 1. • If the value of parameter LOAD changes from 0 to 1, then the value at parameter PV is loaded to the counter as the new CV. • If the value of the reset parameter R is changes from 0 to 1, the current count value is reset to 0.	

Counter data retention after a RUN-STOP-RUN transition or a CPU power cycle

If a run mode session is ended with stop mode or a CPU power cycle and a new run mode session is started, then the counter data stored in the previous run mode session is lost, unless the counter data structure is specified as retentive (CTU, CTD, and CTUD counters).

When you accept the defaults in the call options dialog after you place a counter instruction in the program editor, you are automatically assigned an instance DB which **cannot be made retentive**. To make your counter data retentive, you must either use a global DB or a Multi-instance DB.

Assign a global DB to store counter data as retentive data

This option works regardless of where the counter is placed (OB, FC, or FB).

1. Create a global DB:
   - Double-click "Add new block" from the Project tree
   - Click the data block (DB) icon
   - For the Type, choose global DB
   - If you want to be able to select individual items in this DB as retentive, be sure the symbolic-access-only box is checked.
   - Click OK
2. Add counter structure(s) to the DB:
   - In the new global DB, add a new static tag using one of the counter data types. Be sure to consider the Type you want to use for your Preset and Count values.

Counter Data Type	Corresponding Type for the Preset and Count Values
IEC_Counter	INT
IEC_SCounter	SINT
IEC_DCounter	DINT
IEC_UCounter	UINT
IEC_USCounter	USINT
IEC_UDCounter	UDINT

1. In the "Retain" column, check the box so that this structure will be retentive.
   - Repeat this process to create structures for all the counters that you want to store in this DB. You can either place each counter structure in a unique global DB, or you can place multiple counter structures into the same global DB. You can also place other static tags besides counters in this global DB. Placing multiple counter structures into the same global DB allows you to reduce your overall number of blocks.
   - Rename the counter structures if desired.
2. Open the program block for editing where you want to place a retentive counter (OB, FC, or FB).
3. Place the counter instruction at the desired location.

4. When the call options dialog appears, click the cancel button. You should now see a new counter instruction which has "???" both just above and just below the instruction name.
5. On the top of the new counter instruction, type the name (do not use the helper to browse) of the global DB and counter structure that you created above (example: "Data_block_3.Static_1"). This causes the corresponding preset and count value type to be filled in (example: UInt for an IEC_UCounter structure).

Assign a multi-instance DB to store counter data as retentive data

This option only works if you place the counter in an FB.

This option depends upon whether the FB was created as symbolic access only. Once the FB has been created, you cannot change the checkbox for "Symbolic access only"; it must be chosen correctly when the FB is created, on the first screen after selecting "Add new block" from the tree. To see how this box is configured for an existing FB, right-click on the FB in the Project tree, choose properties, and then choose attributes.

If the FB was created with the "Symbolic access only" box checked:

1. Open the FB for edit.
2. Place the counter instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the Multi instance icon. The Multi Instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the counter if desired.
5. Click OK. The counter instruction appears in the editor with type INT for the preset and count values, and the IEC_COUNTER structure appears in the FB Interface under Static.
6. If desired, change the type in the counter instruction from INT to one of the other types. The counter structure will change correspondingly.

Type shown in counter instruction (for preset and count values)	Corresponding structure Type shown in FB interface
INT	IEC_Counter
SINT	IEC_SCounter
DINT	IEC_DCounter
UINT	IEC_UCounter
USINT	IEC_USCounter
UDINT	IEC_UDCounter

1. If necessary, open the FB interface editor (may have to click on the small arrow to expand the view).
2. Under Static, locate the counter structure that was just created for you.
3. In the Retain column for this counter structure, change the selection to "Retain". Whenever this FB is called later from another program block, an instance DB will be created with this interface definition which contains the counter structure marked as retentive.

If the FB was created with the "Symbolic access only" box *not* checked:

1. Open the FB for edit.
2. Place the counter instruction at the desired location in the FB.
3. When the Call options dialog appears, click on the multi instance icon. The multi instance option is only available if the instruction is being placed into an FB.
4. In the Call options dialog, rename the counter if desired.
5. Click OK. The counter instruction appears in the editor with type INT for the preset and count value, and the IEC_COUNTER structure appears in the FB Interface under Static.
6. If desired, change the type in the counter instruction from INT to one of the other types. The counter structure will change correspondingly.

Type shown in counter instruction (for preset and count values)	Corresponding structure Type shown in FB interface
INT	IEC_Counter
SINT	IEC_SCounter
DINT	IEC_DCounter
UINT	IEC_UCounter
USINT	IEC_USCounter
UDINT	IEC_UDCounter

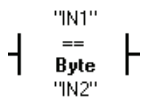
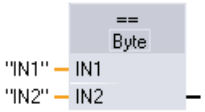
1. Open the block that will use this FB.
2. Place this FB at the desired location. Doing so results in the creation of an instance data block for this FB.
3. Open the instance data block created when you placed the FB in the editor.
4. Under Static, locate the counter structure of interest. In the Retain column for this counter structure, check the box to make this structure retentive.

7.4 Compare

7.4.1 Compare

Table 7- 27 Compare instructions

LAD	FBD	SCL	Description

		`out := in1 = in2;`
		or
		`IF in1 = in2`
		`THEN out := 1;`
		`ELSE out := 0;`
		`END_IF;`
			Compares two values of the same data type. When the LAD contact comparison is TRUE, then the contact is activated. When the FBD box comparison is TRUE, then the box output is TRUE.

[1] For LAD and FBD: Click the instruction name (such as "==") to change the comparison type from the drop-down list. Click the "???" and select data type from the drop-down list.

Table 7- 28 Data types for the parameters

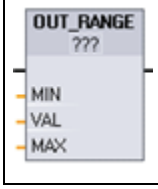
Parameter	Data type	Description
IN1, IN2	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, String, Char, Time, DTL, Constant	Values to compare

Table 7- 29 Comparison descriptions

Relation type	The comparison is true if ...
=	IN1 is equal to IN2
<>	IN1 is not equal to IN2
>=	IN1 is greater than or equal to IN2
<=	IN1 is less than or equal to IN2
>	IN1 is greater than IN2
<	IN1 is less than IN2

7.4.2 In-range and Out-of-range instructions

Table 7- 30 In Range and Out of Range instructions

LAD / FBD	SCL	Description
	`out := IN_RANGE(min, val, max);`	Tests whether an input value is in or out of a specified value range. If the comparison is TRUE, then the box output is TRUE.
	`out := OUT_RANGE(min, val, max);`	

[1] For LAD and FBD: Click the "???" and select the data type from the drop-down list.

Table 7- 31 Data types for the parameters

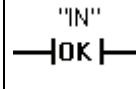
Parameter	Data type[1]	Description
MIN, VAL, MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Comparator inputs

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

- The IN_RANGE comparison is true if: MIN <= VAL <= MAX
- The OUT_RANGE comparison is true if: VAL < MIN or VAL > MAX

7.4.3 OK and Not OK instructions

Table 7- 32 OK and Not OK instructions

LAD	FBD	SCL	Description
		Not available	Tests whether an input data reference is a valid real number according to IEEE specification 754.
		Not available	

[1] For LAD and FBD: When the LAD contact is TRUE, the contact is activated and passes power flow. When the FBD box is TRUE, then the box output is TRUE.

Table 7- 33 Data types for the parameter

Parameter	Data type	Description
IN	Real, LReal	Input data

Table 7- 34 Operation

Instruction	The Real number test is TRUE if:
OK	The input value is a valid real number [1]
NOT_OK	The input value is not a valid real number [1]

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

See also

EN and ENO for LAD, FBD and SCL (Page 163)

7.5 Math

7.5.1 Calculate instruction

Table 7- 35 CALCULATE instruction

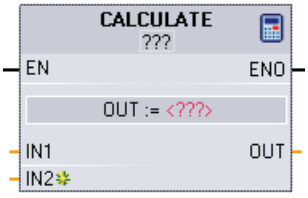
LAD / FBD	SCL	Description
	Use the standard SCL math expressions to create the equation.	The CALCULATE instruction lets you create a math function that operates on inputs (IN1, IN2, .. INn) and produces the result at OUT, according to the equation that you define. • Select a data type first. All inputs and the output must be the same data type. • To add another input, click the icon at the last input.

Table 7- 36 Data types for the parameters

Parameter	Data type[1]
IN1, IN2, ..INn	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord

[1] The IN and OUT parameters must be the same data type (with implicit conversions of the input parameters). For example: A SINT value for an input would be converted to an INT or a REAL value if OUT is an INT or REAL

Click the calculator icon to open the dialog and define your math function. You enter your equation as inputs (such as IN1 and IN2) and operations. When you click "OK" to save the function, the dialog automatically creates the inputs for the CALCULATE instruction.

An example and a list of possible math operations you can include is shown at the bottom of the editor.

Note

You also must create an input for any constants in your function. The constant value would then be entered in the associated input for the CALCULATE instruction.

By entering constants as inputs, you can copy the CALCULATE instruction to other locations in your user program without having to change the function. You then can change the values or tags of the inputs for the instruction without modifying the function.

When CALCULATE is executed and all the individual operations in the calculation complete successfully, then the ENO = 1. Otherwise, ENO = 0.

7.5.2 Add, subtract, multiply and divide instructions

Table 7- 37 Add, subtract, multiply and divide instructions

LAD / FBD	SCL	Description

	`out := in1 + in2;`
	`out := in1 - in2;`
	`out := in1 * in2;`
	`out := in1 / in2;`
		• ADD: Addition (IN1 + IN2 = OUT)
		• SUB: Subtraction (IN1 - IN2 = OUT)
		• MUL: Multiplication (IN1 * IN2 = OUT)
		• DIV: Division (IN1 / IN2 = OUT)
		An Integer division operation truncates the fractional part of the quotient to produce an integer output.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 38 Data types for the parameters (LAD and FBD)

Parameter	Data type[1]	Description
IN1, IN2	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Math operation inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Math operation output

[1] Parameters IN1, IN2, and OUTmust be the same data type.

To add an ADD or MUL input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

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

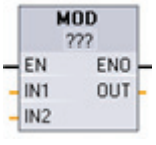
When enabled (EN = 1), the math instruction performs the specified operation on the input values (IN1 and IN2) and stores the result in the memory address specified by the output parameter (OUT). After the successful completion of the operation, the instruction sets ENO = 1.

Table 7- 39 ENO status

ENO	Description
1	No error
0	The Math operation result value would be outside the valid number range of the data type selected. The least significant part of the result that fits in the destination size is returned.
0	Division by 0 (IN2 = 0): The result is undefined and zero is returned.
0	Real/LReal: If one of the input values is NaN (not a number) then NaN is returned.
0	ADD Real/LReal: If both IN values are INF with different signs, this is an illegal operation and NaN is returned.
0	SUB Real/LReal: If both IN values are INF with the same sign, this is an illegal operation and NaN is returned.
0	MUL Real/LReal: If one IN value is zero and the other is INF, this is an illegal operation and NaN is returned.
0	DIV Real/LReal: If both IN values are zero or INF, this is an illegal operation and NaN is returned.

7.5.3 Modulo instruction

Table 7- 40 MOD instruction

LAD / FBD	SCL	Description
	`out := in1 MOD in2;`	You can use the MOD instruction to return the remainder of an integer division operation. The value at the IN1 input is divided by the value at the IN2 input and the remainder is returned at the OUT output.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.