

INTELART

Programmable Logic Controller I4 Series

System Manual

IEC 61131 Compliant
Version 1.4
07/2020

Warning notice system

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

DANGER

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

WARNING

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

NOTICE

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

TIP

indicates that an additional contextual information about a particular element or subject.

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

Qualified Personnel

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

Proper use of Intelart products

Note the following:

WARNING

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

Disclaimer of Liability

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

Preface

Purpose of the manual

The I4 PLCs is a line of programmable logic controllers (PLCs) that can control a variety of automation applications. Compact design, low cost, and a powerful instruction set make the I4 PLC a perfect solution for controlling a wide variety of applications. The I4 models and the Windows-based programming tool give you the flexibility you need to solve your automation problems.

This manual provides information about installing and programming the I4 PLC and is designed for engineers, programmers, installers, and electricians who have a general knowledge of programmable logic controllers.

Required basic knowledge

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

Service and support

In addition to our documentation, we offer our technical expertise on the Internet on the customer support forum (<https://www.intelart.ir/forum>).

Contact your Intelart distributor or sales office for assistance in answering any technical questions, for training, or for ordering I4 products. Because your sales representatives are technically trained and have the most specific knowledge about your operations, process and industry, as well as about the individual Intelart products that you are using, they can provide the fastest and most efficient answers to any problems you might encounter.

Documentation and information

The I4 PLC system manual provides specific information about the operation, programming and the specifications for the complete I4 PLC product family.

If you are a first-time user of I4 PLC, you should read the entire I4 Programmable Controller System Manual. If you are an experienced user, refer to the table of contents or index to find specific information.

The other appendices provide additional reference information, such as descriptions of the error codes, descriptions of the Special Memory (S) area, part numbers for ordering I4 PLC family equipment.

An Overview of IEC 61131-3

The International Electrotechnical Commission (IEC) is the international body that sets global standards for electrical, electronic, and other related technologies.

Various region-specific technology certification bodies derive their standards from IEC. IEC 61131 is the standard for programmable controllers.

It has ten parts covering general information, equipment requirement, user guidelines, communication protocols, safety, fuzzy control programming, and many other aspects regarding programmable controllers.

The third part of IEC 61131 defines the programming languages used for programmable controllers. It was published in December 1993 by IEC, and the current edition (third edition) was released in 2013.

Before IEC 61131-3, different vendors had various programming languages and interoperability was nonexistent. An engineer who knew how to program with one vendor's software had to learn new programming of another vendor to work with the PLC. The different software life-cycle stages are specification, design, implementation, testing, installation, and maintenance, which were heterogeneous for different vendors.



IEC 61131-3 defined a minimum set, the basic programming elements, syntactic and semantic rules for a minimum set, the basic programming elements, syntactic and semantic rules for a programming language used for programmable controllers. The advantages of IEC 61131-3 are:

- Improved interoperability of programming languages
- Higher programming efficiency
- Reduced errors
- Improved reusability
- Modularization
- Implementation of modern software techniques
- Increased user efficiency

This made IEC 61131-3 widely accepted by users and vendors globally and has become the standard for programming and configuring industrial control devices. The standards are also evolving according to the needs of the industry and inefficiencies in earlier editions. A noted enhancement in the subsequent editions is the addition and improvement of support for Object-Oriented Programming (OOP), including classes, methods, interfaces, and namespaces.

IEC 61131-3 defines the basic structure and elements of all programming languages for programming controllers. This allows PLCs to be programmed using multiple languages. Note that software used to program PLCs from one vendor cannot typically be used to program PLCs from another vendor.

This is due to the differences in addressing schemes, task scan rates, array sizes, string lengths, and file formats. Users can utilize the five programming languages to program PLCs. The features of IEC 61131-3 that aids this are mentioned below.

Standard Data Types

IEC defined a standard set of data types with uniform across all the programming software compliant with IEC 61131-3. The standard has defined how to interpret the contents of the variable. Only one type of operation is allowed for a particular data type. For example, mathematical operations can be done only on numerical data types and not on bit-patterns.

Derived Data Types

High-level, PC-oriented programming languages offer derived data types that users can define according to their needs. This gives more flexibility and versatility to programming languages. IEC 61131-3 supports derived data types like Fields and structures that allow for efficient organization and grouping of data. This allows for the use of data in a secure manner.

Program Organization Units (POU)

Functions and function blocks are the most common POUs for programming. Recurrent tasks can be bundled as functions or function blocks that can be called when they are required.

This division of sub-tasks of the whole program makes programming and verifying written programs easier. It will be legible and coherent, opposed to the mess when such POUs are not available while programming.

Data Encapsulation

The third edition of IEC 61131-3 supports object-oriented programming. This is enabled by the capability of data encapsulation. It is the practice of bundling data with the functions that use the data. Classes are the most common use case of data encapsulation, widely used in high-level object-oriented computer languages.

With this, all the POUs have only local data and cannot be manipulated by other parts of the program. This avoids overwriting data errors.

Data-Exchange Interfaces

It's necessary to have POUs and data encapsulation to have a robust programming language and define the data-exchange interfaces. The data types and the scope of each data type in different POUs must be well defined. IEC 61131-3 has standardized the data exchange interfaces for programming languages for logic controllers.

Symbolic Functions & Function Blocks

Using the IEC 61131-3 standards, programs can be written in a way that is address and module independent. This enables writing functions and function blocks that are independent of target systems. The logic takes precedence over the specific implementation. This allows users to write reusable programs that can be appropriated for various systems.

Standard Syntax and Semantics

Syntaxes and semantics make up a high-level computer language. IEC 61131-3 has standardized them for all programmable controller languages. The commands and instructions would be the same across various programming languages.

This reduces the training required by engineers if they have to work with PLCs from multiple vendors. This additional feature enhances the reusability of the programs.

Language Extensions

IEC 61131-3 has no intention of reducing the development of new PLC languages but only to standardize the languages. The standard allows proprietary function blocks to be programmed in non-IEC 61131-3 languages such as C++.

There are PLC vendors and dedicated software vendors that write reusable programs in higher-level PC-oriented languages. These are then ported to use with a specific device. PLC vendors can also enhance and provide extensions to the programming languages that are IEC 61131-3 compliant.

All these features help with using multiple languages for the same PLC according to the comfort of the user.

This standard helped unify, to an extent, the heterogeneous and fragmented programming landscape for PLCs.

 TIP

All I4 PLCs and their programming software is designed based on IEC61131 standard. For more information you can refer the IEC 6131-3 documentation.

Table of Contents

1	Product Overview.....	1
1.	I4 PLC.....	2
2.	I4 PLC Expansion Modules.....	3
3.	Intelart Studio Programming Package	4
3.1	Computer Requirements	4
3.2	Installing Intelart Studio	4
4.	Communications Options.....	4
2	Getting Started.....	5
1.	Connecting the I4 PLC.....	6
1.1	Connecting Power to the I4 PLC	6
1.2	Connecting the Programming Cable.....	6
1.3	Starting Intelart Studio	6
1.4	Establishing Communications with the I4 PLC	7
2.	Creating a Sample Program	8
2.1	Opening the Program Editor.....	10
2.2	How to Program.....	10
2.3	Saving the Sample Project	11
3.	Downloading the Sample Program	12
4.	Placing the I4 PLC in RUN Mode.....	12
5.	Easy-to-use tools	13
5.1	Inserting instructions into your user program.....	13
5.2	Inserting Instructions from the “Quick Access” Toolbar	14
5.3	Adding inputs or outputs to a LAD or FBD instruction	14
5.4	Selecting a version for an instruction.....	15
5.5	Modifying the appearance and configuration of Intelart Studio.....	16
5.6	Changing the operating mode of the CPU.....	16
5.7	Modifying the Hardware Configuration of CPU and Expansion Modules.....	17
5.8	Mapping Module Tags	17
5.9	Importing license files	18
3	Installing the I4 PLC.....	20
1.	Guidelines for Installing I4 PLC Devices.....	21
1.1	Separate the I4 PLC Devices from Heat, High Voltage, and Electrical Noise.....	21
1.2	Provide Adequate Clearance for Cooling and Wiring	21
2.	Installing and removing the I4 PLC Modules.....	22
2.1	Prerequisites.....	22
2.2	Mounting Dimensions	22
2.3	Installing a CPU or Expansion Module	23
2.4	Removing a CPU or Expansion Module	23
3.	Guidelines for Grounding and Wiring.....	24
3.1	Prerequisites.....	24
3.2	Guidelines for Isolation	24
3.3	Guidelines for Grounding the I4 PLC.....	24
3.4	Guidelines for Wiring the I4 PLC	25

3.5	Guidelines for Inductive Loads	25
3.6	Guidelines for Lamp Loads.....	26
4	PLC Concepts.....	28
1.	Execution of the user program.....	28
1.1	Operating modes of the CPU.....	30
1.2	Processing the scan cycle in RUN mode.....	31
1.3	Organization blocks (OBs).....	32
1.4	CPU memory.....	34
1.5	Time of day clock.....	34
1.6	Configuring the outputs on a RUN-to-STOP transition	34
2.	Data storage, memory areas, I/O and addressing	36
2.1	Accessing the data of the I4 PLC	36
2.2	Configuring the I/O in the CPU and I/O modules	38
3.	Processing of analog values	40
4.	Data types.....	41
4.1	Bool, Byte, Word, DWord and LWord data types.....	42
4.2	Integer data types.....	42
4.3	Floating-point real data types	43
4.4	Time and Date data types	43
4.5	Character and String data types	44
4.6	Array data type	46
4.7	Data structure data type	46
4.8	User data type	46
4.9	Pointer data types.....	47
5	Device Configuration.....	49
1.	Inserting a CPU.....	51
2.	Adding modules to the configuration.....	52
3.	Configuring the operation of the CPU	52
4.	Configuring the parameters of the modules	53
4.1	Assigning Internet Protocol (IP) addresses	54
6	Programming Concepts	57
1.	Guidelines for designing a PLC system	58
2.	Structuring your user program	59
2.1	Choosing the type of structure for your user program	59
3.	Using blocks to structure your program	60
3.1	Organization block (OB)	61
3.2	Function (FC).....	62
3.3	Function block (FB)	62
4.	Understanding data consistency	64
5.	Programming language.....	64
5.1	Ladder logic (LAD).....	64
5.2	Function Block Diagram (FBD).....	65
5.3	EN and ENO for LAD and FBD.....	65
6.	Protection.....	66

6.1	Access protection for the CPU.....	66
6.2	Program blocks protection	67
6.3	Copy protection	67
6.4	Downloading a compiler binary output file	68
7.	Downloading the elements of your program	68
7.1	Transfer Program to SD Card.....	69
8.	Uploading from the CPU	69
9.	Monitoring and testing the program	69
9.1	Monitor and modify data in the CPU.....	69
9.2	Watch and force list.....	70
9.3	Cross reference to show usage	70
9.4	Call structure to examine the calling hierarchy	70
7	Basic Instructions.....	71
1.	Bit logic	74
1.1	Bit logic contacts and coils	74
1.2	Set and reset instructions	76
1.3	Positive and negative edge instructions	77
2.	Word logic operations	79
2.1	AND, OR, and XOR instructions.....	79
2.2	Invert instruction	79
2.3	Shift and Rotate.....	80
2.4	Rotate instructions.....	80
3.	Comparison.....	81
3.1	Compare.....	81
3.2	In-range and Out-of-range instructions.....	82
4.	Math.....	82
4.1	Add, subtract, multiply and divide instructions.....	82
4.2	Modulo instruction	83
	General exponentiation instruction.....	84
4.3	Absolute value instruction.....	84
4.4	Increment and decrement instructions.....	84
4.5	Floating-point math instructions.....	85
5.	Timer and Counter	85
5.1	Timers	85
5.2	Counters.....	89
6.	Moving and conversion	92
6.1	Move instructions.....	92
6.2	Accessing data by array indexing	93
6.3	Convert instruction.....	93
6.4	BCD conversion instructions.....	94
6.5	Round, ceiling, floor and truncate instructions.....	94
6.6	Swap instruction	95
6.7	Serialize instruction	96
6.8	Deserialize instruction	96

7.	Program Control.....	97
7.1	FOR statement.....	97
7.2	WHILE statement.....	98
7.3	IF statement.....	98
7.4	RET execution control instruction.....	99
8.	Selection.....	100
8.1	Select.....	100
8.2	Get maximum and minimum.....	101
8.3	Limit instruction.....	101
8.4	Multiplex instruction.....	102
8.5	Check for nullity.....	102
8.6	Check for array.....	102
8.7	Get array length.....	103
9.	Time.....	103
9.1	Time add and subtract.....	103
9.2	Time multiplication and division.....	104
9.3	Time of day addition and subtraction time.....	104
9.4	Date addition and subtraction time.....	105
9.5	Date subtraction.....	105
9.6	Time of day subtraction.....	106
9.7	Date and time subtraction.....	106
9.8	Time concatenation.....	107
10.	Character and string.....	107
10.1	String data overview.....	107
10.2	String operation instructions.....	107
8	System Instructions.....	113
1.	Memory management.....	114
1.1	RWW_NVMEM instruction.....	114
2.	System Time Management.....	114
2.1	GET_SYS_DT instruction.....	114
2.2	SET_SYS_DT instruction.....	115
2.3	SYS_TICK instruction.....	115
3.	Comm ports management.....	115
3.1	SET_SYS_IP.....	115
9	Communication Instructions.....	117
1.	RS-232 interface.....	118
2.	RS-485 interface.....	118
2.1	Bias resistors.....	118
2.2	Termination resistors.....	118
2.3	Shielding and grounding considerations.....	118
2.4	Cable requirements.....	119
3.	Controller Area Network (CAN) interface.....	119
4.	Ethernet interface.....	119
4.1	Modbus TCP/IP.....	120

4.2	EtherCAT	120
4.3	Ethernet/IP	120
4.4	PROFINET	120
5.	Programming instructions	120
5.1	Serial	120
6.	Modbus communication	123
6.1	Overview of Modbus RTU and TCP communication	123
6.2	Modbus RTU	124
6.3	Modbus TCP.....	129
10	IEC 61131-3 Solutions	134
1.	CMD_MONITOR instruction.....	135
2.	STACK_INT FB instruction	135
3.	LAG1 FB instruction.....	136
4.	DELAY FB instruction	136
5.	AVERAGE FB instruction.....	137
6.	INTEGRAL FB instruction	137
7.	DERIVATIVE FB instruction.....	138
8.	HYSTERESIS FB instruction	138
9.	LIMITS_ALARM FB instruction	139
10.	ANALOG_MONITOR FB instruction	139
11.	IEC_PID FB instruction	140
12.	RAMP FB instruction.....	141
13.	TRANSFER FB instruction.....	141
11	Monitor and Control Instructions	143
1.	Designing Digital Controllers.....	144
1.1	Process Characteristics and Control	144
1.2	Feedforward Control.....	147
1.3	Multi-Loop Controls	147
1.4	Structure and Mode of Operation of the PID Control	150
1.5	Signal Processing in the Setpoint Branch.....	153
1.6	Signal Processing in the PID Controller.....	153
2.	Configuring and Starting the Standard PID Control	154
2.1	Defining the Control Task	154
2.2	Type of Actuator	155
2.3	Generating the Control Project Configuration.....	156
2.4	The Sampling Time CYCLE.....	157
2.5	How the Standard PID Control is Called.....	158
2.6	Range of Values and Signal Adaptation (Normalization).....	158
3.	Signal Processing in the Setpoint/Process Variable Channels and PID Controller Functions	158
3.1	Average Value Generator (AVG_GEN)	158
3.2	Rate of Change Alarm Generator (CHG_ALM)	159
3.3	Cycle Time Calculator (CYC_TM)	160
3.4	Filtering Signal Function (DEADBAND).....	161
3.5	Unsigned Int to Signed Int Encoder (ENCODER).....	162

3.6	First In First Out (FIFO)	162
3.7	Asymmetric Hysteresis Generator (HYST_GEN)	163
3.8	Damping the Process Variable (LAG1_GEN)	163
3.9	Monitoring a Process Variable Limits (LIM_ALM)	165
3.10	Loop Scheduler (LP_SCHED)	166
3.11	Manual Value Generator (MAN_GEN)	168
3.12	Normalize (NORM)	169
3.13	Standard PID (PID_STD)	170
3.14	PWM Signal Generator (PWM_GEN)	175
3.15	PID Tuner by Relay Method (RELAY_TUNE)	176
3.16	Ramp Soak (RMP_GEN)	177
3.17	Limiting the Rate of Change of a Value (ROC_GEN)	183
3.18	Scale (SCALE)	184
3.19	Gain Scheduling (SCH_GEN)	184
3.20	Scale With Parameters (SCP_NORM)	185
3.21	PID Self Tuner (SELF_TUNE)	186
3.22	Extracting the Square Root Normalization (SQRT_NORM)	190
3.23	Stack Collection (STACK)	191
3.24	Three Step Signal Generator (THREE_STEP_GEN)	191
3.25	Weighing System (WEIGH)	192
12	Technology Instructions	193
1.	Temperature Control	194
1.1	Temperature Control by TEMP_CONTROLLER	194
1.2	Temperature Control Optimizer (TEMP_OPT)	197
13	Online and Diagnostic Tools	198
1.	Status LEDs	199
1.1	Status LEDs on a CPU	199
2.	Going online and connecting to a CPU	199
3.	Displaying the status of the CPU	200
4.	Setting the date and time of day	200
5.	Displaying or setting CPU configuration	200
6.	Resetting to factory settings	200
6.1	Procedure	201
6.2	Result	201
7.	CPU operator toolbar for the online CPU	201
8.	Monitoring and modifying values in the CPU	201
8.1	Going online to monitor the values in the CPU	201
8.2	Displaying status in the program editor	202
8.3	Using a watch table to monitor and modify values in the CPU	202
9.	Recovery from a lost password	203
10.	Runtime Exceptions	203
11.	CPU registers	205

1

Product Overview

The I4 PLC series can control a wide variety of devices to support your automation needs.

The I4 PLC monitors inputs and changes outputs as controlled by the user program, which can include Boolean logic, counting, timing, complex math operations, and communications with other intelligent devices. The compact design, flexible configuration, and powerful instruction set combine to make the I4 PLC a perfect solution for controlling a wide variety of applications.

1. I4 PLC

The I4 PLC combines a microprocessor, an integrated power supply, input circuits, and output circuits in a compact housing to create a powerful PLC. See Figure 1-1. After you have downloaded your program, the I4 PLC contains the logic required to monitor and control the input and output devices in your application.

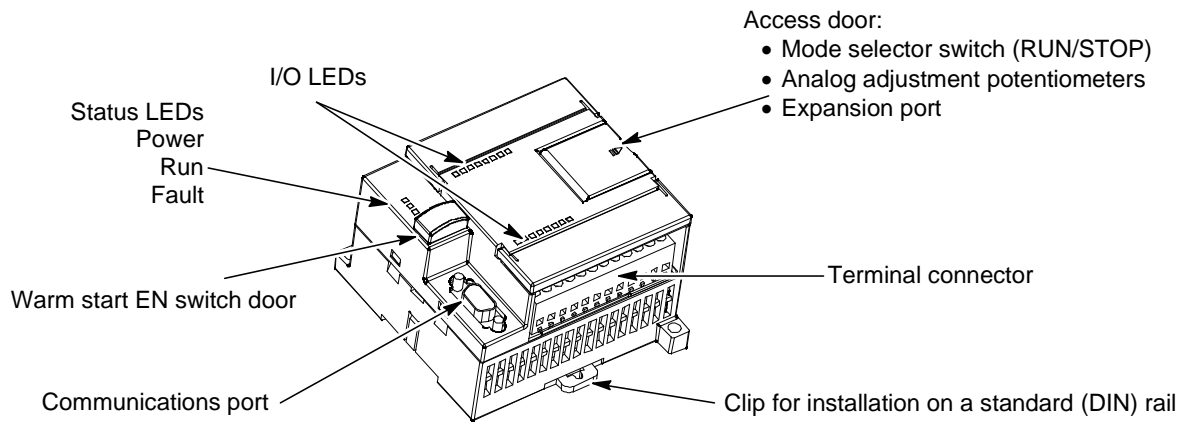


Figure 1-1 I4 PLC

Intelart provides different I4 CPU models with a diversity of features and capabilities that help you create effective solutions for your varied applications. Table 1-1 briefly compares some of the features of the CPU. For detailed information about a specific CPU, see Appendix A.

Table 1-1 Comparison of the I4 CPU Models

Feature	CP300	CP301	CP310
Physical size (mm)	90 x 96 x 61		121 x 96 x 61
Load memory	4 MB		
Application memory	80 KB		160 KB
Retentive memory	72 bytes		4 KB
Memory			
• M	8 KB		32 KB
• I	2 KB		8 KB
• Q	2 KB		8 KB
• G	8 KB		32 KB
Permanent Memory	16 KB		
Memory backup	5 Years typical		
Supported modules	31		63
I/O integrated in CPU	Yes, 6 DI, 8 DQ	Yes, 6 DI, 4 DQ	Yes, 13 DI, 16 DQ
Fast counters	3		
PWM	3, 1 KHz	0	3, 1 KHz
Pulse train	3	0	3
Frequency out	3	0	3
Analog POTs	2		
Real-time clock	Built-in		
Communications ports	1 RS-232, 1 RS-485		1 RS-232, 1 RS-485, 1 Ethernet
Programming port	USB		Ethernet
Floating-point math	Yes		
Processor	Arm, Cortex M4		Arm, Cortex M7
Bit operation	10 µs		4 µs
CPU startup modes	Cold start, Warm start		
Configuration / programming Software	Intelart Studio		

2. I4 PLC Expansion Modules

To better solve your application requirements, the I4 PLC family includes a wide variety of expansion modules. You can use these expansion modules to add additional functionality to the I4 PLC. Table 1-2 provides a list of the expansion modules that are currently available. For detailed information about a specific module, see Appendix A.

Table 1-2 I4 PLC Expansion Modules

Module	Inputs	Outputs
Digital Modules		
IM300	8 (Sink/Source)	8 (NPN)
IM301	8 (Sink/Source)	4 (Relay)
IM310	16 (Sink/Source)	0
IM320	0	16 (NPN)
IM330	0	8 (Relay)
Analog Modules		
IM341	2 (0-24 mA, 0-10 V)	2 (0-24 mA, 0-10 V)
IM342	4 (0-24 mA, 0-10 V)	0
IM350	1 (DC, RTD, RES, Thermocouple)	1 (0-24 mA, 0-10 V)
IM351	2 (DC, RTD, RES, Thermocouple)	2 (0-24 mA, 0-10 V)
IM360	1 (Loadcell)	1 (0-24 mA, 0-10 V)
IM361	2 (Loadcell)	1 (0-24 mA, 0-10 V)

3. Intelart Studio Programming Package

The Intelart Studio programming package provides a user-friendly environment to develop, edit, and monitor the logic needed to control your application. Intelart Studio provides editors for convenience and efficiency in developing the control program for your application. To help you find the information you need, Intelart Studio provides an extensive online help forum and technical support.

3.1 Computer Requirements

Your computer or programming device should meet the following minimum requirements:

- Operating system: Windows 7 or higher
- Processor type: Pentium 4 or higher
- At least 500 MB of free hard disk space
- At least 1 GB of free RAM space

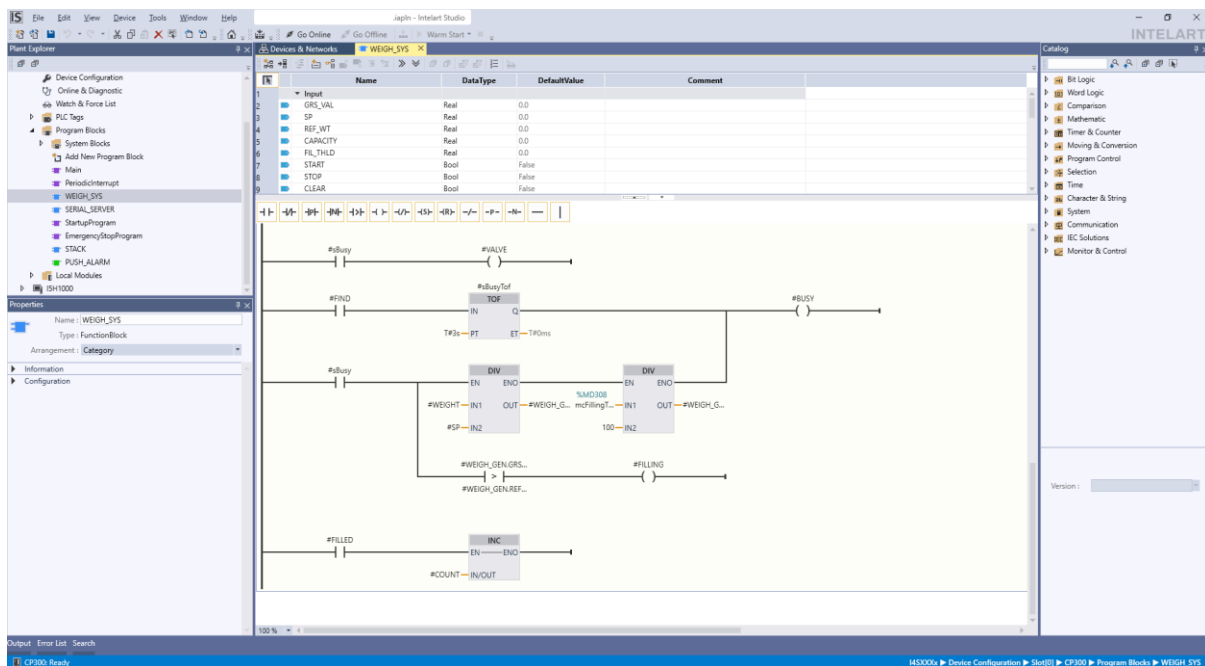


Figure 1-2 Intelart Studio

3.2 Installing Intelart Studio

You can download setup file from the following URL:

<https://intelart.ir/products/software-products/intelart-studio>

After running the setup file, the installation wizard starts automatically and prompts you through the installation process. Note that some prerequisites may be installed based on your operating system installed runtime packages.

4. Communications Options

Intelart provides the most common and economical method of connecting your computer to the I4 PLC. Based on the type of the CPU you can use an ethernet cable or a USB B 2.0 cable.

2

Getting Started

Intelart Studio makes it easy for you to program your I4 PLC. In just a few short steps using a simple example, you can learn how to connect, program, and run your I4 PLC.

All you need for this example is a USB (or ethernet) cable, an I4 PLC, and a programming device running the Intelart Studio programming software.


1. Connecting the I4 PLC

Connecting your I4 PLC is easy. For this example, you only need to connect power to your I4 PLC and then connect the communications cable (USB or Ethernet) between your programming device and the I4 PLC.

1.1 Connecting Power to the I4 PLC

The first step is to connect the I4 PLC to a power source. Figure 2-1 shows the wiring connections for a DC model of the I4 PLC.

Before you install or remove any electrical device, ensure that the power to that equipment has been turned off. Always follow appropriate safety precautions and ensure that power to the I4 PLC is disabled before attempting to install or remove the I4 PLC.

 WARNING
Attempts to install or wire the I4 PLC or related equipment with power applied could cause electric shock or faulty operation of equipment. Failure to disable all power to the I4 PLC and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment. Always follow appropriate safety precautions and ensure that power to the I4 PLC is disabled before attempting to install or remove the I4 PLC or related equipment.

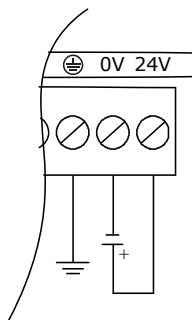


Figure 2-1 Connecting Power to the I4 PLC

1.2 Connecting the Programming Cable

According to its structure, each CPU can be connected to a computer with one of the common cables such as USB or ethernet. To know the type of program cable of your device, refer to Table 1-1 or specific technical sheet of that device.

1.3 Starting Intelart Studio

Click on the Intelart Studio icon and then Create New Plant button to create a new project. Figure 2-4 shows a new project.

Notice the Plant Explorer. You can use the icons on the plant explorer to open elements of the Intelart Studio project.

Click on the Devices & Networks in the plant explorer to display the plant devices. You use this editor to manage all devices and set up the communications for Intelart Studio.

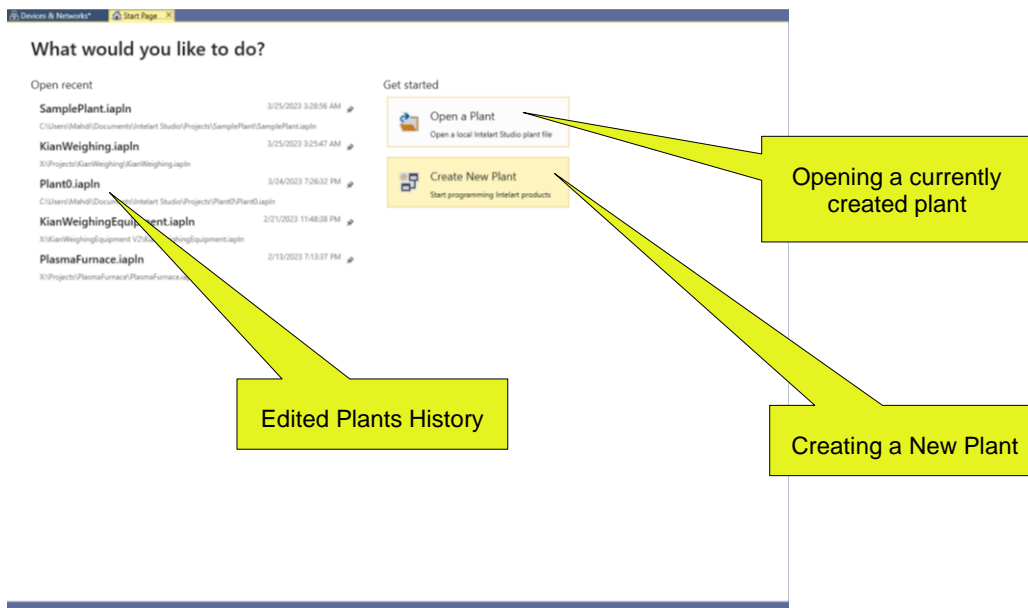


Figure 2-2 Start Page

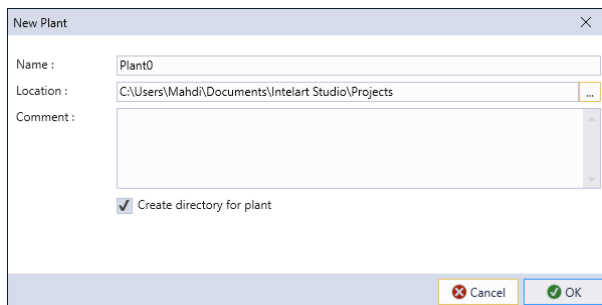


Figure 2-3 New Plant Dialog

1.4 Establishing Communications with the I4 PLC

Click on the program port (PG) on the device element in the Devices & Networks editor. By selecting the PG port, its configuration appears in Properties pane. After you set the PG parameters (IP or COM port name) with the device, you are ready to Go Online to device. If Intelart Studio does not find your I4 PLC, a Programmer Configuration dialog will be appeared in order to change the PG port parameters or search for available devices on a network. If you are using an Ethernet port for programming, then a dialog with IP setting field and search network tool will be appeared. For USB ports you must know the COM port name and select a COM port from available detected connected COM ports from dialog.

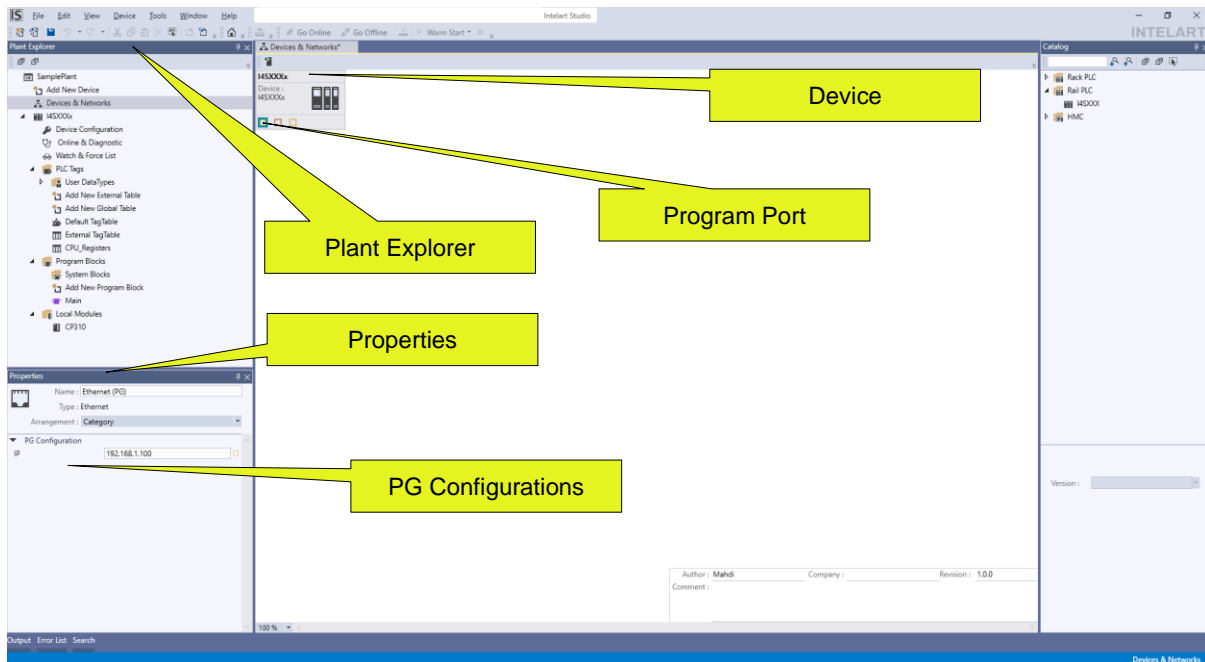


Figure 2-4 New Intelart Studio Project

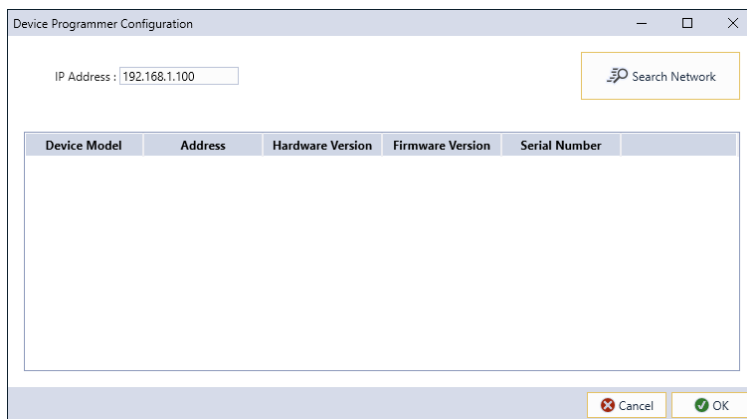
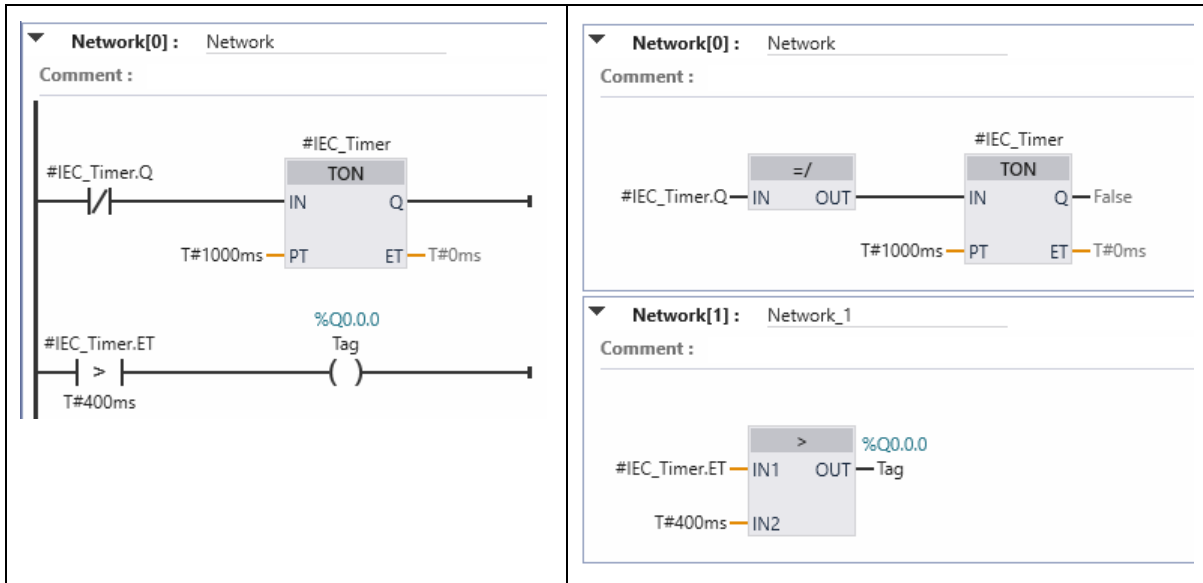


Figure 2-5 Programmer Configuration dialog

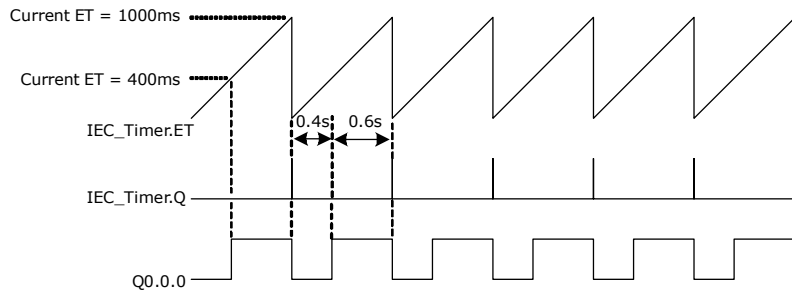
2. Creating a Sample Program

Entering this example of a control program will help you understand how easy it is to use Intelart Studio. This program uses six instructions in three networks to create a very simple, self-starting timer that resets itself.

For this example, you use the Ladder (LAD) editor to enter the instructions for the program. The following example shows the complete program in both LAD and Function Block Diagram (FBD). The timing diagram shows the operation of the program.



Timing Diagram



2.1 Opening the Program Editor

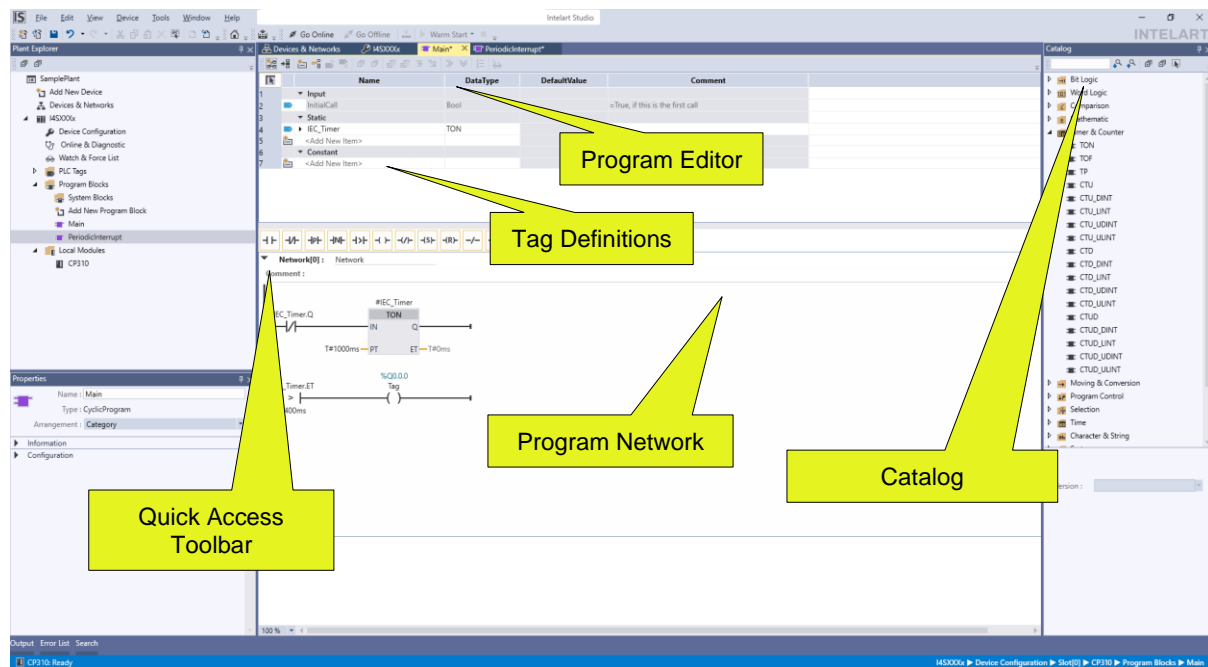


Figure 2-6 A ladder program block

Double-click on a Program Block icon in plant explorer tree to open the program editor. See Figure 2-6.

Notice the catalog pane and the program editor. You use the catalog to insert the programming instructions into the networks of the program editor by dragging and dropping the instructions from the catalog to the networks or by double click on an instruction to place it on current editing network. When a network goes to editing mode, its background gets highlight in order to distinguish the current editing network.

The toolbar icons provide shortcuts to the common instructions in catalog.

After you enter and save the program, you can download the program to the I4 PLC.

2.2 How to Program

When #IEC_Timer.Q is off (0), this contact turns on and provides power flow to start the timer. To enter the contact for #IEC_Timer.Q:

- 1- In the plant explorer, double-click on the Main program block in the Program Blocks folder in order to load its editor.
- 2- Either double-click the Bit Logic icon in the catalog or click on the triangle sign (▶) to display the bit logic instructions.
- 3- Select the Normally Closed contact.
- 4- Hold down the left mouse button and drag the contact onto the first cell of network.
- 5- Double-click on the “???” above the contact and enter the following tag: #IEC_Timer.Q
- 6- Press the Enter key to submit the tag for the contact.

To enter the timer instruction for TON:

- 1- Create a TON tag named “IEC_Timer “in the tag list in the top area of the program editor.
- 2- Double-click the Timer & Counter icon in catalog to display the timer instructions.
- 3- Select the TON (On-Delay Timer).
- 4- Hold down the left mouse button and drag the timer onto the next cell in network.
- 5- Click on the “???” above the timer box (Instance) and enter the following created tag name in step 1: IEC_Timer

- 6- Press the Enter key or click outside the editing box to submit the timer instance name and the double click on the PT input of the timer in order to set the preset time.
- 7- Enter the following value for the preset time (PT): T#1000ms
- 8- Press the Enter key to submit the value.

When the timer elapsed time (ET) for IEC_Timer is greater than 400 milliseconds, (or 0.4 seconds), the contact provides power flow to turn on output DQ0 of the I4 PLC. To enter the Compare instruction:

- 1- Double-click the Comparison icon to display the compare instructions. Select the > instruction (Greater Than).
- 2- Hold down the left mouse button and drag the compare instruction onto the second row of network.
- 3- Click on the contact in order to select it. Notice the properties pane. You can customize some properties for any selected element in all editors. Change the OperationDataType in the properties pane to Time.
- 4- Double-click on the “???” above the contact and enter the elapsed time for the timer: #IEC_Timer.ET.
- 5- Press the Enter key to submit the timer instance internal tag and Double-click on the “???” below the contact.
- 6- Enter the following value to be compared with the timer value: T#400ms
- 7- Press the enter key to submit the value.

To enter the instruction for turning on output DQ0:

- 1- Double-click the Bit Logic icon to display the bit logic instructions and select the output coil.
- 2- Hold down the left mouse button and drag the coil onto the next network.
- 3- Double-click on the “???” above the coil and enter the following address: %Q0.0.0
- 4- Press the Enter key to enter the tag for the coil. If a tag with the above address already exists, the editor will assign that tag to the argument of the instruction elsewhere it will create a tag with the specified address and then assigns the created tag to the instruction argument.

When the timer reaches the preset value (ET=1000ms) and turns the timer bit on, the contact for IEC_Timer turns on. Because the timer is enabled by a Normally Closed contact for #IEC_Timer.Q, changing the state of #IEC_Timer.Q from off (0) to on (1) resets the timer.

2.3 Saving the Sample Project

After entering the set of instructions, you have finished entering the program. When you save the program, you create a new project file that includes the I4 PLC CPU type and other parameters. A file with the “.bak” extension will be created next to the saved file contains the previous state of the created plant from last save.

To save a currently opened plant in another place:

- 1- Select the File > Save As menu command from the menu bar.
- 2- Choose one of the two saving methods.
- 3- Click OK to save the plant.

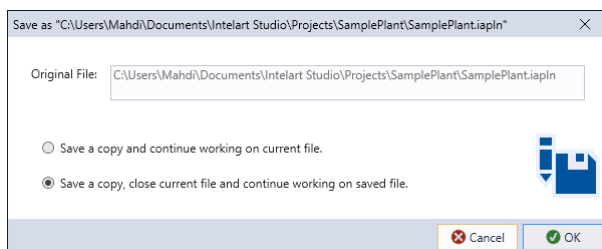


Figure 2-7 Saving as the Example Program

TIP

You can make a backup of current state of the editing plant by File > Make Backup or press Ctrl+B simultaneously.

After saving the project, you can download the program to the I4 PLC.

3. Downloading the Sample Program

TIP

Each Intelart Studio project is associated with a CPU type. If the target device type does not match the CPU to which you are connected, Intelart Studio indicates a mismatch message in Output pane.

- 1- Click on the Go Online toolbar button or press Ctrl+K simultaneously in order to connect the device. See figure 2-8.
- 2- Click the Compile and Download icon on the toolbar or select the relevant menu command in the Device menu or press F6 to compile the current device and download the compiled program.

If your I4 PLC is in RUN mode, a dialog box prompts you to place the I4 PLC in STOP mode. Click Yes to place the I4 PLC into STOP mode.

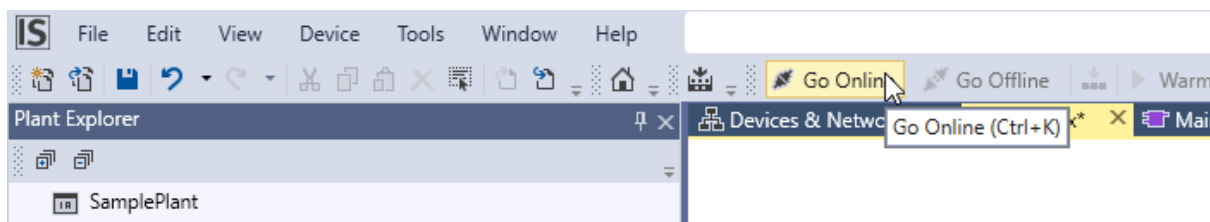


Figure 2-8 Go Online toolbar button

4. Placing the I4 PLC in RUN Mode

For Intelart Studio to place the I4 PLC CPU in RUN mode, the mode switch of the I4 PLC will be override by the Intelart Studio. When you place the I4 PLC in RUN mode, the I4 PLC executes the program:

- 1- Click the Warm Start (or Cold Start) icon on the toolbar or select the Device > Warm Start Device menu command or press F5.
- 2- Click OK in the dialog to change the operating mode of the I4 PLC.

When the I4 PLC goes to RUN mode, the output LED for DQ0 turns on and off as the I4 PLC executes the program.

Congratulations! You have just completed your first I4 PLC program.

You can monitor the program by selecting the Monitor Continuously toolbar button in each program editor.

Intelart Studio displays the values for the instructions. To stop the program, place the I4 PLC in STOP mode by clicking the Stop toolbar button or by selecting the Device > Stop Device menu command or press Shift+F5.

WARNING

When you disconnect the computer from device, its operating mode will be changed to the selected switch state.

5. Easy-to-use tools

5.1 Inserting instructions into your user program

Intelart Studio provides a Catalog pane that brings up relevant elements when an editor opens. In program editor the instructions are grouped according to their functionality.

To create your program, you drag instructions from the catalog pane onto a network.

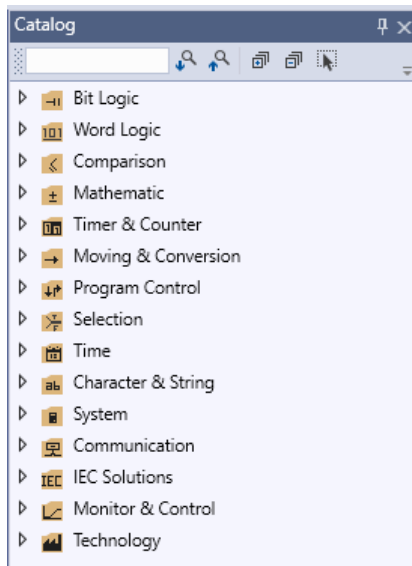


Figure 2-9 Catalog Pane

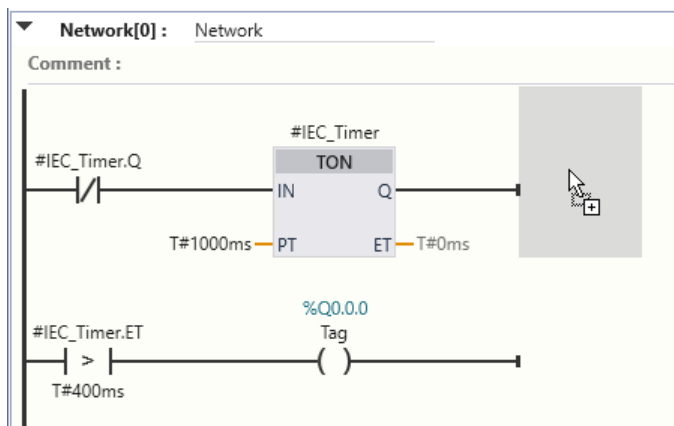


Figure 2-10 Dragging an Instruction on a LAD Network

5.2 Inserting Instructions from the “Quick Access” Toolbar

Intelart Studio provides a “Quick Access” toolbar to give you quick access to the instructions that frequently use in programming. Simply click the icon for the instruction to insert it into your network!

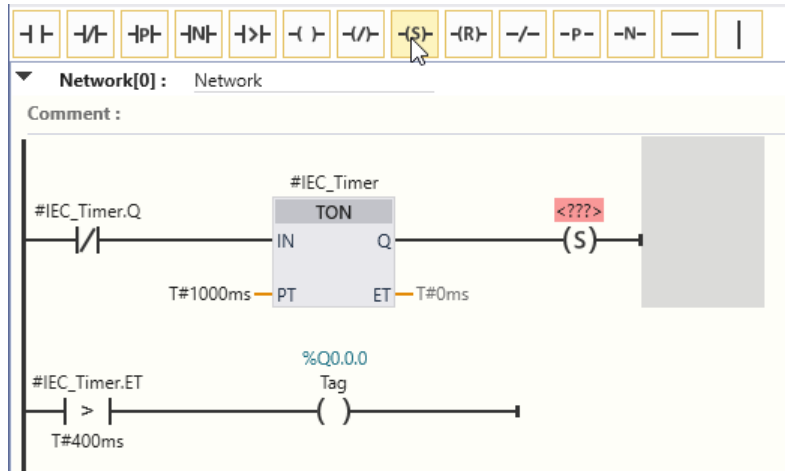


Figure 2-11 Quick Access Toolbar

5.3 Adding inputs or outputs to a LAD or FBD instruction

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

To add or remove last input or output, select that instruction by mouse then click the "▲" or "▼" icon in the Properties pane. You can also specify the count by entering a number in the relevant editing box.

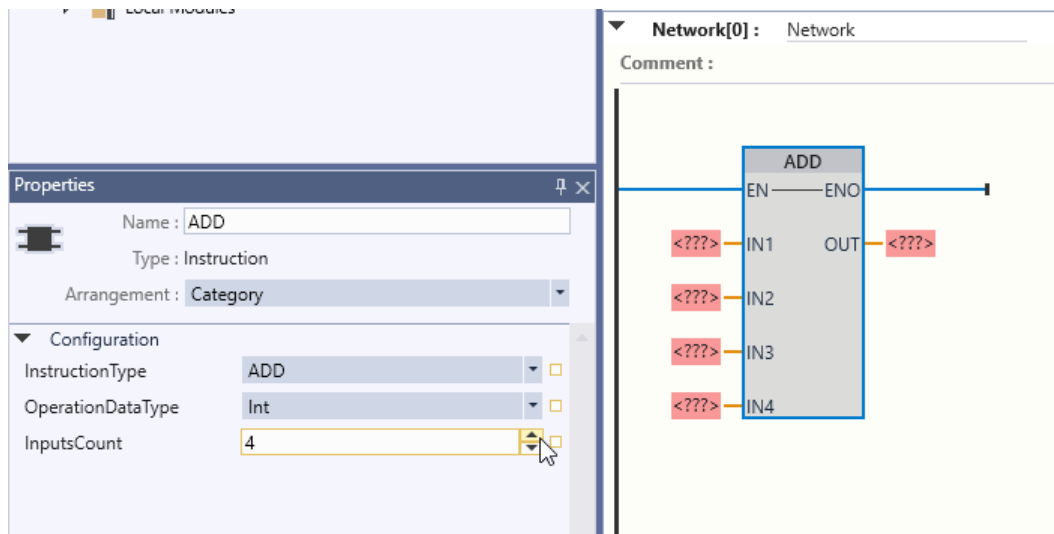


Figure 2-12 Increase or Decrease Inputs or Outputs of a Specific Instruction

5.4 Selecting a version for an instruction

The development and release cycles for certain sets of instructions (such as Modbus, PID and motion) have created multiple released versions for these instructions. Also, some instructions have multiple functions. To help ensure compatibility and migration with older projects, or use of another functionality of an instruction Intelart Studio allows you to choose which version of instruction to insert into your user program.

Click the icon on the instruction Catalog to enable the version selection box of the instruction. To change the version of the instruction, select the appropriate version from the drop-down list.

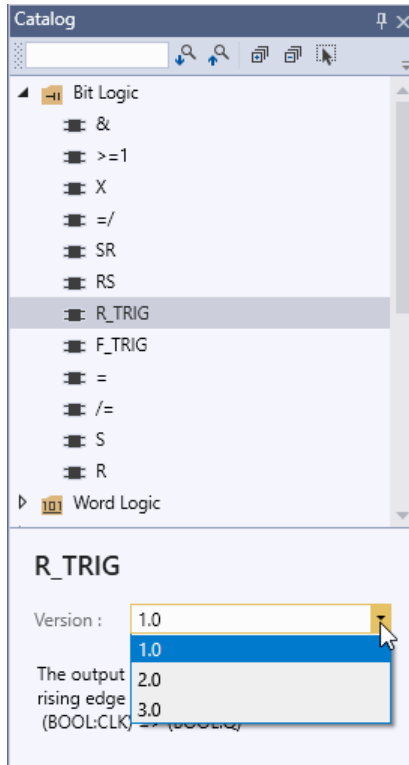


Figure 2-13 Choosing different version of an Instruction

5.5 Modifying the appearance and configuration of Intelart Studio

You can select a variety of settings, such as the appearance of the interface, or other settings for more customization of your work bench.

Select the “ Options ” command from the “Tools” menu to change these settings.

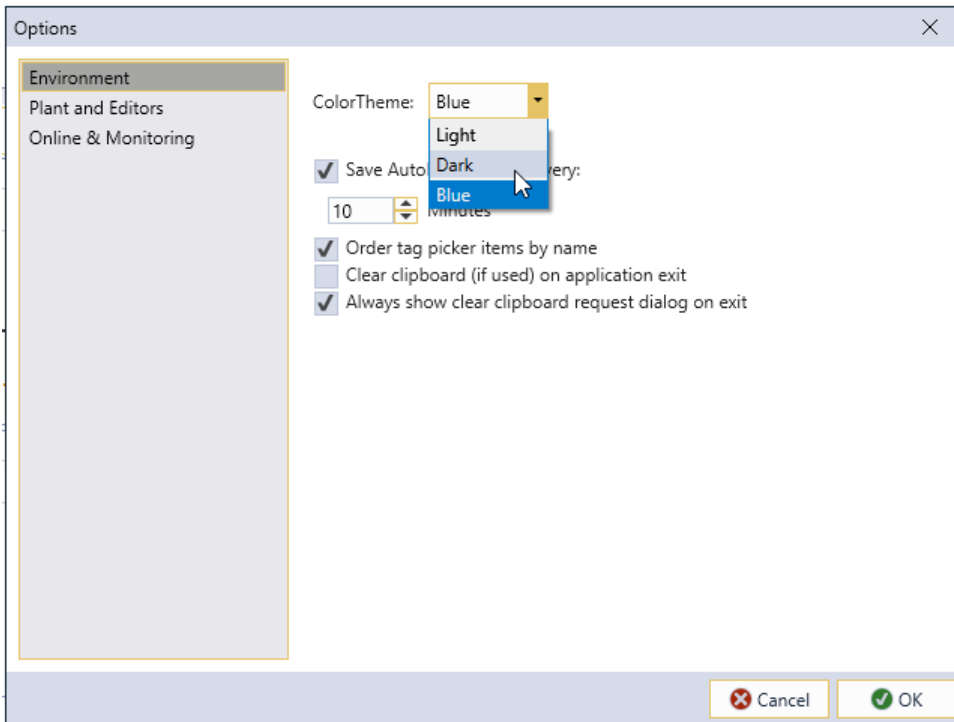


Figure 2-14 Options Dialog

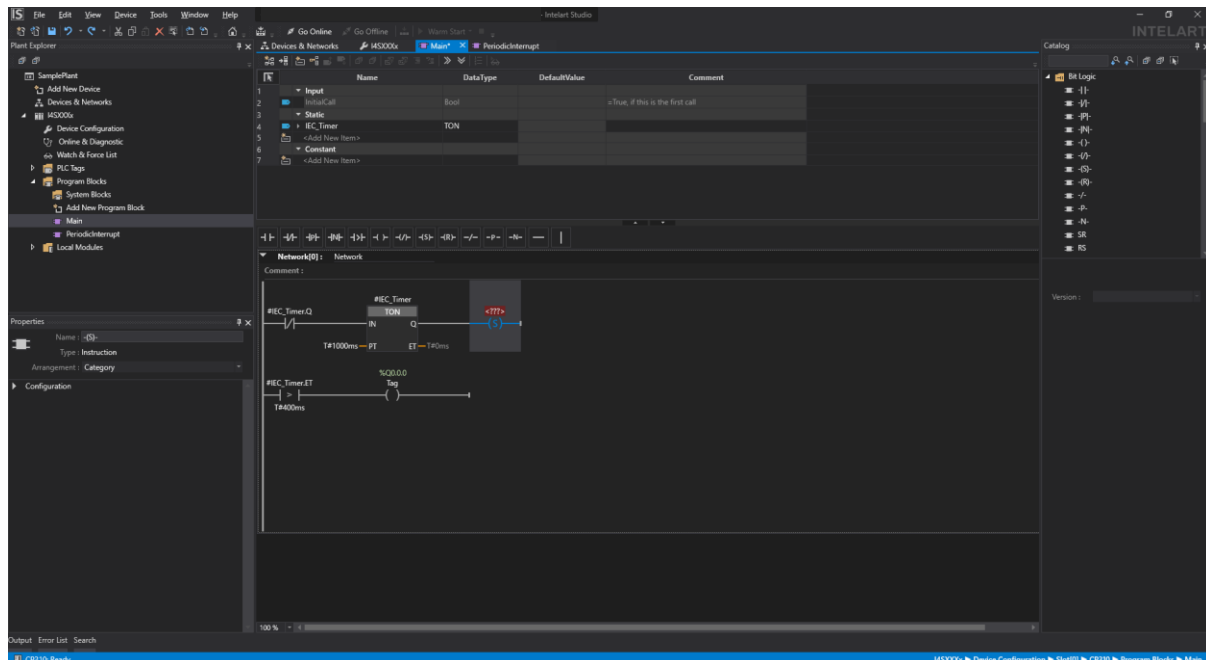


Figure 2-15 Dark Theme Applied to Intelart Studio

5.6 Changing the operating mode of the CPU

Use the “Warm Start” (or “Cold Start”) and “Stop” toolbar buttons to change the operating mode of the CPU.

When you configure the CPU in the device configuration, you configure the start-up behavior in the “Online & Diagnostic” of the CPU.

The “Online and diagnostics” also provides an operator panel for changing the operating mode of the online CPU. To use the CPU operator panel, you must be connected online to the CPU. The “Status” task card displays an operator panel that shows the operating mode of the online CPU. The operator panel also allows you to change the operating mode of the online CPU or other system configurations such as password or Emergency Stop trigger. Also, you can change RTC or upgrade the firmware of device by this panel.

5.7 Modifying the Hardware Configuration of CPU and Expansion Modules

Double-click “Device Configuration” in plant explorer then you see a schematic of CPU and its expansion modules. By clicking on a module, its configuration parameters appear in Properties pane.

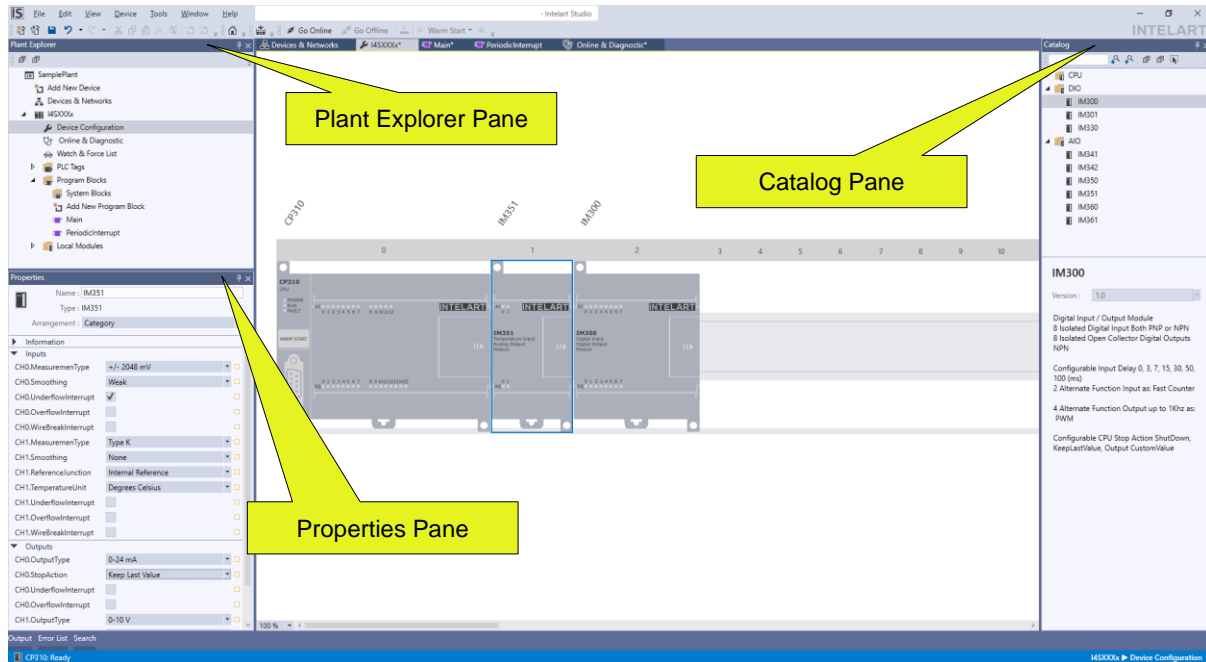


Figure 2-16 Device Configuration Editor

By dragging and dropping the expansion modules from the catalog to the rail or by double click on an expansion module to place it on the current rail.

5.8 Mapping Module Tags

When you install a CPU or an expansion module, its hardware tags will be accessible by double-click on the module schematic in “Device Configuration” editor or by finding it in “Local Modules” folder in Plant Explorer pane and double-click on it. You should do the following:

- 1- Select all tags you need in the list
- 2- Click on “Map Tags” toolbar button
- 3- Select an external tag table in the dialog in order to place the mapped tags
- 4- Click on Ok to Intelart Studio create the selected tags in the list

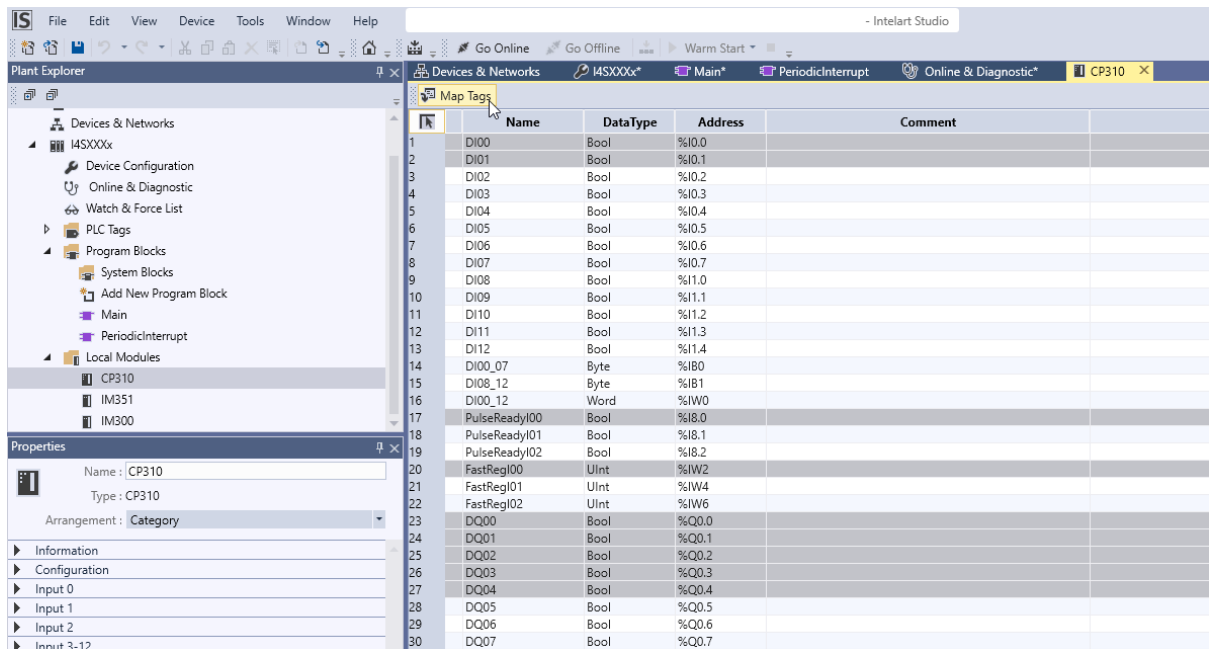


Figure 2-17 A module tags list

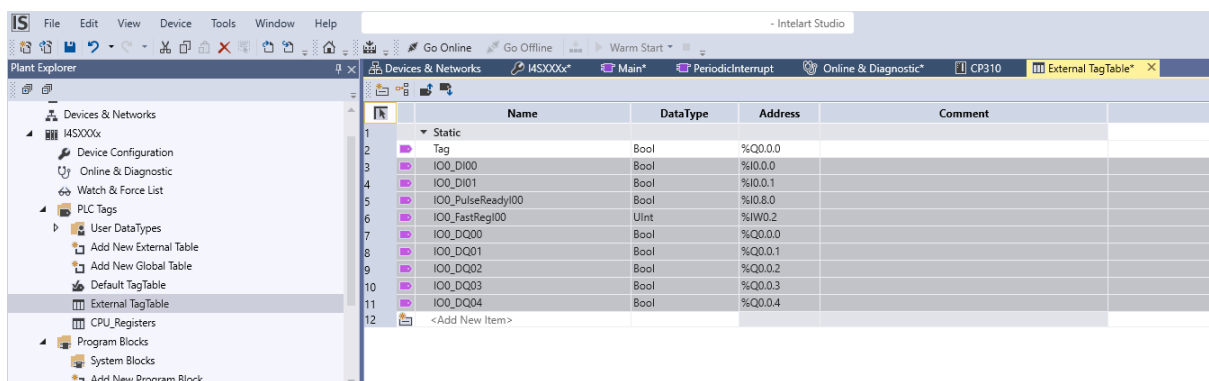


Figure 2-18 Created tags in the external tag table

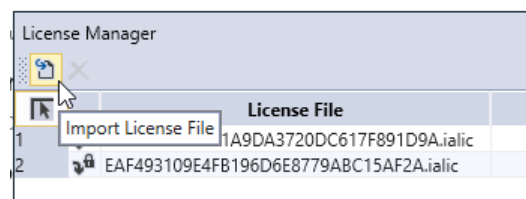
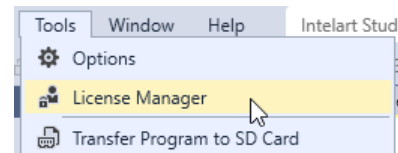
NOTICE

Addressing of I and Q area tags is a bit different from M area tags. All I and Q area tags address starts by a number that indicates the physical address of that area.

5.9 Importing license files

You can include license files (.ialic) in license manager by following steps:

- 1- Open License Manager in Tools menu.
- 2- In the opened dialog click on "Import License File" button in toolbar.
- 3- Select the file and then press "Close" button.



3

Installing the I4 PLC

The I4 PLC equipment is designed to be easy to install. You can use the mounting holes to attach the modules to a panel, or you can use the built-in clips to mount the modules onto a standard (DIN) rail. The small size of the I4 PLC allows you to make efficient use of space.

This chapter provides guidelines for installing and wiring your I4 PLC system.

1. Guidelines for Installing I4 PLC Devices

You can install an I4 PLC either on a panel or on a standard rail, and you can orient the I4 PLC either horizontally or vertically.

⚠ WARNING

The I4 PLC devices are Open Type Controllers. It is required that you install the I4 PLC in a housing, cabinet, or electric control room. Entry to the housing, cabinet, or electric control room should be limited to authorized personnel.

Failure to follow these installation requirements could result in death or serious injury to personnel, and/or damage to equipment.

Always follow these requirements when installing I4 PLC devices.

1.1 Separate the I4 PLC Devices from Heat, High Voltage, and Electrical Noise

As a general rule for laying out the devices of your system, always separate the devices that generate high voltage and high electrical noise from the low-voltage, logic-type devices such as the I4 PLC.

When configuring the layout of the I4 PLC inside your panel, consider the heat-generating devices and locate the electronic-type devices in the cooler areas of your cabinet. Operating any electronic device in a high-temperature environment will reduce the time to failure.

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

1.2 Provide Adequate Clearance for Cooling and Wiring

I4 PLC devices are designed for natural convection cooling. For proper cooling, you must provide a clearance of at least 25 mm above and below the devices. Also, allow at least 75 mm of depth.

⚠ WARNING

For vertical mounting, the maximum allowable ambient temperature is reduced by 10 degrees C. Mount the I4 PLC CPU below any expansion modules.

When planning your layout for the I4 PLC system, allow enough clearance for the wiring and communications cable connections. For additional flexibility in configuring the layout of the I4 PLC system, use the I/O expansion cable.

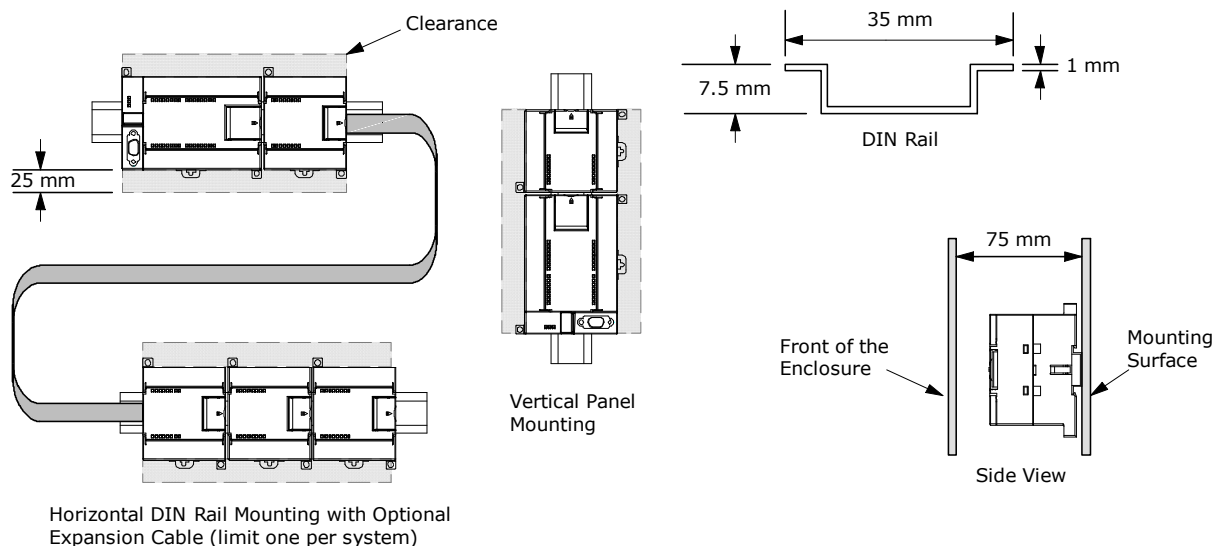


Figure 3-1 Mounting Methods, Orientation, and Clearance

2. Installing and removing the I4 PLC Modules

The I4 PLC can be easily installed on a standard DIN rail or on a panel.

2.1 Prerequisites

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

⚠ WARNING

Attempts to install or remove I4 PLC or related equipment with the power applied could cause electric shock or faulty operation of equipment.
 Failure to disable all power to the I4 PLC and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.
 Always follow appropriate safety precautions and ensure that power to the I4 PLC is disabled before attempting to install or remove I4 PLC CPUs or related equipment.

Always ensure that whenever you replace or install an I4 PLC device you use the correct module or equivalent device.

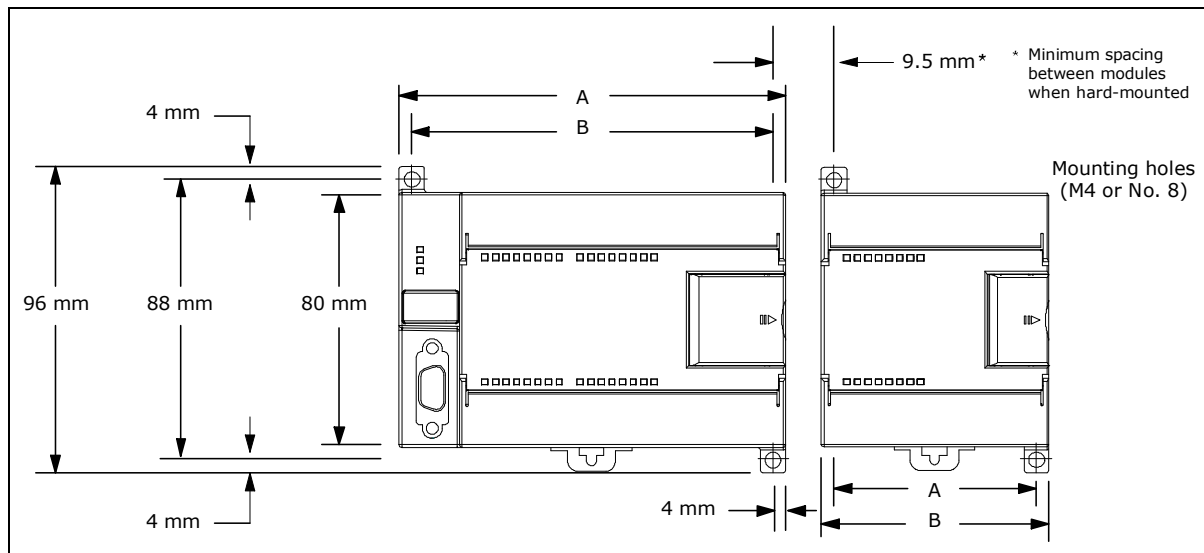
⚠ WARNING

If you install an incorrect module, the program in the I4 PLC will generate an IO exception.
 Failure to replace an I4 PLC device with the same model, orientation, or order could result in death or serious injury to personnel, and/or damage to equipment.
 Replace an I4 PLC device with the same model, and be sure to orient and position it correctly.

2.2 Mounting Dimensions

The I4 PLC CPUs and expansion modules include mounting holes to facilitate installation on panels. Refer to Table 3-1 for the mounting dimensions.

Table 3-1 Mounting Dimensions



I4 Module	Width A	Width B
CP300, CP301	90 mm	82 mm
CP310	121 mm	113 mm
IM300, IM301, IM310, IM320, IM330	71.2 mm	63.2 mm
IM341, IM342, IM350, IM351, IM360, IM361	46 mm	38 mm

2.3 Installing a CPU or Expansion Module

Installing the I4 PLC is easy! Just follow these steps.

Panel Mounting

- 1- Locate, drill, and tap the mounting holes (M4 or American Standard number 8), using the dimensions in Table 3-1.
- 2- Secure the module(s) to the panel, using the appropriate screws.
- 3- If you are using an expansion module, connect the expansion module ribbon cable into the expansion port connector under the access door.

DIN Rail Mounting

- 1- Secure the rail to the mounting panel every 75 mm.
- 2- Snap opens the DIN clip (located on the bottom of the module) and hook the back of the module onto the DIN rail.
- 3- If you are using an expansion module, connect the expansion module ribbon cable into the expansion port connector under the access door.
- 4- Rotate the module down to the DIN rail and snap the clip closed. Carefully check that the clip has fastened the module securely onto the rail. To avoid damage to the module, press on the tab of the mounting hole instead of pressing directly on the front of the module.

TIP

Using DIN rail stops could be helpful if your I4 PLC is in an environment with high vibration potential or if the I4 PLC has been installed vertically.
If your system is in a high-vibration environment, then panel-mounting the I4 PLC will provide a greater level of vibration protection.

2.4 Removing a CPU or Expansion Module

To remove an I4 PLC CPU or expansion module, follow these steps:

- 1- Remove power from the I4 PLC.
- 2- Disconnect all the wiring and cabling that is attached to the module.
- 3- If you have expansion modules connected to the unit that you are removing, open the access cover door and disconnect the expansion module ribbon cable from the adjacent modules.
- 4- Unscrew the mounting screws or snap open the DIN clip.
- 5- Remove the module.

3. Guidelines for Grounding and Wiring

Proper grounding and wiring of all electrical equipment is important to help ensure the optimum operation of your system and to provide additional electrical noise protection for your application and the I4 PLC.

3.1 Prerequisites

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

Ensure that you follow all applicable electrical codes when wiring the I4 PLC and related equipment. Install and operate all equipment according to all applicable national and local standards. Contact your local authorities to determine which codes and standards apply to your specific case.

WARNING

Attempts to install or wire the I4 PLC or related equipment with power applied could cause electric shock or faulty operation of equipment. Failure to disable all power to the I4 PLC and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.

Always follow appropriate safety precautions and ensure that power to the I4 PLC is disabled before attempting to install or remove the I4 PLC or related equipment.

Always take safety into consideration as you design the grounding and wiring of your I4 PLC system. Electronic control devices, such as the I4 PLC, can fail and can cause unexpected operation of the equipment that is being controlled or monitored. For this reason, you should implement safeguards that are independent of the I4 PLC to protect against possible personal injury or equipment damage.

WARNING

Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment. Use an emergency stop function, electromechanical overrides, or other redundant safeguards that are independent of the I4 PLC.

3.2 Guidelines for Isolation

I4 PLC AC power supply boundaries and I/O boundaries to AC circuits have been designed and approved to provide safe separation between AC line voltages and low voltage circuits. These boundaries include double or reinforced insulation, or basic plus supplementary insulation, according to various standards. Components which cross these boundaries such as optical couplers, capacitors, transformers, and relays have been approved as providing safe separation.

WARNING

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

Such unexpected high voltages could result in death or serious injury to personnel, and/or damage to equipment.

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

3.3 Guidelines for Grounding the I4 PLC

The best way to ground your application is to ensure that all the common and ground connections of your I4 PLC and related equipment are grounded to a single point. This single point should be connected directly to the earth ground for your system.

For improved electrical noise protection, it is recommended that all DC common returns be connected to the same single-point earth ground. Connect the 24 VDC sensor supply common to earth ground.

All ground wires should be as short as possible and should use a large wire size, such as 2 mm² (14 AWG).

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

3.4 Guidelines for Wiring the I4 PLC

When designing the wiring for your I4 PLC, provide a single disconnect switch that simultaneously removes power from the I4 PLC CPU power supply, from all input circuits, and from all output circuits. Provide overcurrent protection, such as a fuse or circuit breaker, to limit fault currents on supply wiring. You might want to provide additional protection by placing a fuse or other current limit in each output circuit.

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

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

Use the shortest wire possible and ensure that the wire is sized properly to carry the required current. The connector accepts wire sizes from 2 mm² to 0.3 mm² (14 AWG to 22 AWG). Use shielded wires for optimum protection against electrical noise. Typically, grounding the shield at the I4 PLC gives the best results.

When wiring input circuits that are powered by an external power supply, include an overcurrent protection device in that circuit. External protection is not necessary for circuits that are powered by the 24 VDC sensor supply from the I4 PLC because the sensor supply is already current-limited.

To avoid damaging the connector, be careful that you do not over-tighten the screws. The maximum torque for the connector screw is 0.56 N-m (5 inch-pounds).

To help prevent unwanted current flows in your installation, the I4 PLC provides isolation boundaries at certain points. When you plan the wiring for your system, you should consider these isolation boundaries. Refer to Appendix A for the amount of isolation provided and the location of the isolation boundaries. Isolation boundaries rated less than 1500 VAC must not be depended on as safety boundaries.

💡 TIP

For a communications network, the maximum length of the communications cable is 50 m without using a repeater. The communications port on the I4 PLC is non-isolated. Refer to Chapter 7 for more information.

3.5 Guidelines for Inductive Loads

You should equip inductive loads with suppression circuits to limit voltage rise when the control output turns off. Suppression circuits protect your outputs from premature failure due to high inductive switching currents. In addition, suppression circuits limit the electrical noise generated when switching inductive loads.

💡 TIP

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

DC Outputs and Relays That Control DC Loads

The DC outputs have internal protection that is adequate for most applications. Since the relays can be used for either a DC or an AC load, internal protection is not provided.

Figure 3-2 shows a sample suppression circuit for a DC load. In most applications, the addition of a diode (A) across the inductive load is suitable, but if your application requires faster turn-off times, then the addition of a Zener diode (B) is recommended. Be sure to size your Zener diode properly for the amount of current in your output circuit.

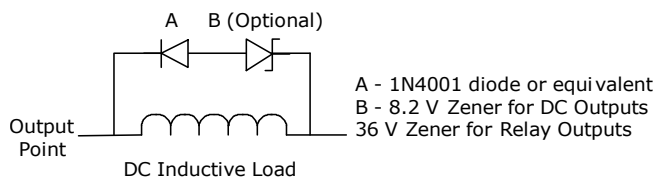


Figure 3-2 Suppression Circuit for a DC Load

AC Outputs and Relays That Control AC Loads

The AC outputs have internal protection that is adequate for most applications. Since the relays can be used for either a DC or an AC load, internal protection is not provided.

Figure 3-3 shows a sample suppression circuit for an AC load. When you use a relay or AC output to switch 115 V/230 VAC loads, place resistor/capacitor networks across the AC load as shown in this figure. You can also use

a metal oxide varistor (MOV) to limit peak voltage. Ensure that the working voltage of the MOV is at least 20% greater than the nominal line voltage.

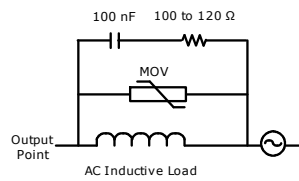


Figure 3-3 Suppression Circuit for an AC Load

⚠ WARNING

When relay expansion modules are used to switch AC inductive loads, the external resistor/capacitor noise suppression circuit must be placed across the AC load to prevent unexpected machine or process operation. See Figure 3-3.

3.6 Guidelines for Lamp Loads

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

4

PLC Concepts

1. Execution of the user program

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

- Organization blocks (OBs) define the structure of the program. Some OBs have predefined behavior and start events, but you can also create OBs with custom start events.
- Functions (FCs) and function blocks (FBs) contain the program code that corresponds to specific tasks or combinations of parameters. Each FC or FB provides a set of input and output parameters for sharing data with the calling block.

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

A function (FC) or a function block (FB) is a block of program code that can be called from an OB or from another FC or FB.

The size of the user program, data, and configuration is limited by the available load memory and application memory in the CPU. There is a specific limit to the number of each individual OB, FC, FB and other programming elements. For more details refer to technical data of each CPU.

Each cycle includes writing the outputs, reading the inputs (when has I/O control), executing the user program instructions, and performing background processing. The cycle is referred to as a scan cycle or scan.

The expansion modules are detected and logged in only upon power-up.

Inserting or removing a module in the central rail under power (hot) is not supported.

Never insert or remove a module from the central rail when the CPU has power.

WARNING

Insertion or removal of an expansion module from the central rail when the CPU has power could cause unpredictable behavior, resulting in damage to equipment and/or injury to personnel. Always ensure that power is removed from the CPU before inserting or removing a module from the central rail.

Under the default configuration, all local digital and analog I/O points are updated synchronously with the scan cycle using an internal memory area called the process image.

The process image contains a snapshot of the physical inputs and outputs (the physical I/O points on the CPU and expansion modules).

The CPU performs the following tasks:

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

- The CPU executes the logic of the user instructions and updates the output values in the process image output area instead of writing to the actual physical outputs.

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

You can specify whether digital and analog I/O points are to be automatically updated and stored in the process image. If you insert a module in the device configuration, its data is located in the process image of the CPU (default). The CPU handles the data exchange between the module and the process image area automatically during the update of the process image.

1.1 Operating modes of the CPU

The CPU has three modes of operation: STOP mode, TRANSIENT TO RUN (STARTUP) mode, RUN mode and TRANSIENT TO STOP mode.

Status LEDs on the front of the CPU indicate the current mode of operation.

- In STOP mode, the CPU is not executing the program. You can download a project.
- In STARTUP mode, the CPU config itself and expansion modules. Then the startup OB (if present) is executed once. Interrupt events are not processed during the startup OB execution.
- In RUN mode, the cyclic program OB is executed repeatedly. Interrupt events can occur and be processed at any point within the RUN mode.

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

A memory reset clears all retentive and non-retentive memory areas, and resets all expansion module configurations. A memory reset does not clear the diagnostics buffer or the saved values in permanent memory.

NOTICE

When you change retain tags or a module configuration and download program to CPU, the retentive values are set to their default values. The next transition to RUN performs a warm start, setting all non-retentive data to their default values and setting all retentive data to their retained values.
--

You can configure the "start mode after POWER ON" setting of the CPU. This configuration item appears under the Options in "Online & Diagnostic" for the CPU under "Start Mode". When power is applied, the CPU performs a sequence of power-up diagnostic checks and system initialization. During system initialization the CPU deletes all non-retentive tag values to the initial values from load memory. The CPU retains retentive tag values and then enters the appropriate operating mode. Certain detected errors prevent the CPU from entering the RUN mode.

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

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

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

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

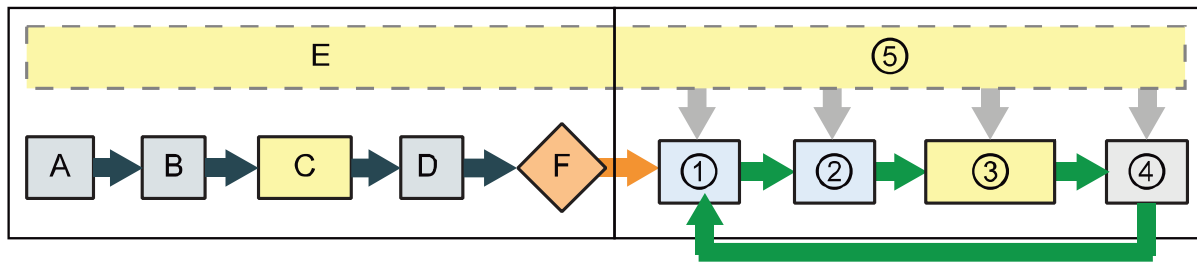


Figure 4-1 Processing cycle steps

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

STARTUP processing

Whenever the operating mode changes from STOP to RUN, the CPU clears the process image inputs, initializes the process image outputs and processes the startup OB. All OBs has a property named "Has IO Control" which specifies whether CPU can update I (image) memory area before the execution of the OB and write Q memory to the physical outputs.

NOTICE
<p>Some OBs executes like a loop such as Cyclic Program, Periodic Interrupt, Time of Day Interrupt. Only one OB can control the I/O simultaneously. Enabling "Has IO Control" property of each loop OB will disable the other loop OBs.</p>

In the startup OB you can determine the validity of retentive data and the time-of-day clock and other system diagnostics information by accessing the special memory tags. You can program instructions inside the startup OB to examine these special tag values and to take appropriate action.

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

Interrupts are queued but not processed during the startup phase

No cycle time monitoring is performed during the startup phase

1.2 Processing the scan cycle in RUN mode

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

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

The following steps will be executed on each OB:

- The scan cycle starts by reading the current values of the digital and analog inputs from the CPU and expansion modules and then writing these values to the process image when the OB has I/O control.

- After reading the inputs, the user program is executed from the first instruction through the end instruction. This includes all the program cycle OB plus all their associated FCs and FBs. The program cycle OB are executed in order according to its priority and execution condition.
- If the OB has I/O control, the scan cycle ends by preparing the current values of the digital and analog outputs from the process image and then writing them to the physical outputs of the CPU and expansion modules.

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

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

1.3 Organization blocks (OBs)

OBs control the execution of the user program. Specific events in the CPU or a hardware event trigger the execution of an organization block. OBs cannot call each other or be called from an FC or FB. Only a start event, such as a diagnostic interrupt or a time interval, can start the execution of an OB. The CPU handles OBs according to their respective priority classes, with higher priority OBs executed before lower priority OBs. The lowest and highest priority class may vary for individual CPUs. In order to see the priority classes for a CPU refer to its technical data.

OBs control the following operations:

- Cyclic Program OBs execute cyclically while the CPU is in RUN mode. The main block of the program is a Cyclic Program OB. This is where you place the instructions that control your program and where you call additional user blocks. Only one Cyclic Program OB is allowed and always executes repeatedly. The Main OB is the default. Other program cycle OBs must be created by user.
- Startup OB executes one time when the operating mode of the CPU changes from STOP to RUN, including powering up in the RUN mode and in commanded STOP-to-RUN transitions. After completion, the main "Cyclic Program" OB will begin executing. Only one startup OB is allowed.
- Periodic interrupt OBs execute at a specified interval. A periodic interrupt OB will interrupt cyclic program execution at user defined intervals, such as every 2 seconds. You can configure up a limited number of these kind of OB. Refer to technical data of a CPU to see the maximum count of such OBs.
- Hardware interrupt OBs execute when the relevant hardware event occurs, including over range or under range of analog inputs. A hardware interrupt OB will interrupt normal cyclic program execution in reaction to a signal from a hardware event. You define the events in the properties of the hardware configuration. One OB is allowed for Hardware interrupt OB.
- Time of Day interrupt OB executes at every second. A Time-of-Day interrupt OB will interrupt cyclic program execution at one second intervals. You can configure only one Time of Day interrupt OB.
- Stop Program OB executes one time when the operating mode of the CPU changes from RUN to STOP.
- Emergency Stop Program OB executes cyclically when a hardware emergency stop trigger occurs. Emergency Stop Program will be executed and all other OBs will be stopped until its hardware trigger is enabled.

The cyclic program executes once during each program cycle (or scan). During the cyclic program, the CPU, reads the inputs, executes program and writes the outputs. The cyclic program event is required and is always enabled. You cannot delete cyclic program OB otherwise the program will not be compiled.

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

Understanding Hardware Interrupt OB

Analog (local) or other expansion modules are capable of detecting and reporting diagnostic errors. The occurrence or removal of any of several different diagnostic error conditions results in a diagnostic error event. The following are some of diagnostic errors that are supported:

- No user power
- High limit exceeded

- Low limit exceeded
- Wire break

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

1.4 CPU memory

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

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

1.4.1 Retentive memory

Data loss after power failure can be avoided by marking certain data as retentive in any "Global Tag Table". To see how much is available for a specified CPU, refer to its technical data.

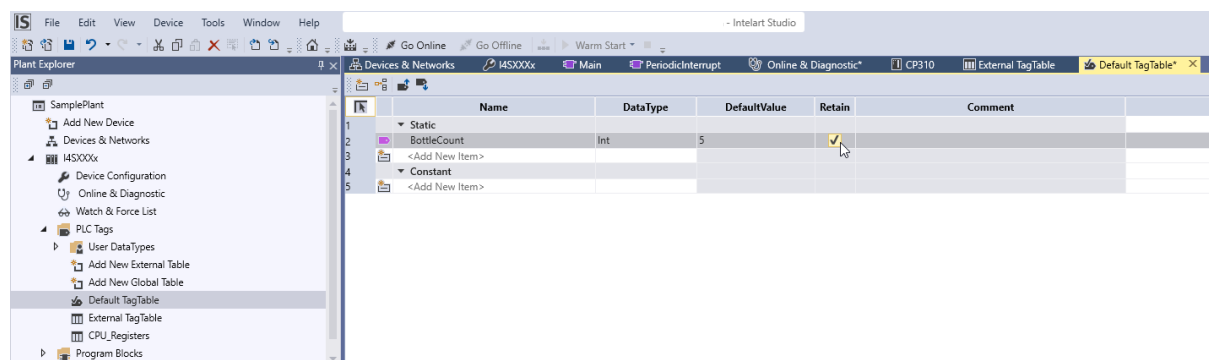


Figure 4-2 Mark a tag as retain

1.5 Time of day clock

The CPU supports a time-of-day clock. A built-in lithium battery supplies the energy required to keep the clock running during times when the CPU is powered down. The battery does not discharge while the CPU has power. Typically, the battery has sufficient charge to keep the clock and retentive area running for typically 5 years.

To utilize the time-of-day clock, you must set it. Timestamps such as those for diagnostic information, data log files, and data log entries are based on the system time. You set the time of day from the "Set time" function in the "Online & diagnostics" view of the online CPU. Intelart Studio then calculates the system time from the time you set plus or minus the Windows operating system offset from UTC (Coordinated Universal Time). Setting the time of day to the current local time produces a system time of UTC if your Windows operating system settings for time zone and daylight savings time correspond to your locale.

I4 PLC includes instructions to read and write the system time (GET_SYS_DT and SET_SYS_DT), to read the date time and to set the date time.

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

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

- Substituting a specified output value (default): You enter a substitute value for each output (channel) of that CPU or expansion module. The default substitute value for digital output channels is OFF, and the default substitute value for analog output channels is 0.
- Freezing the outputs to remain in last state: The outputs retain their current value at the time of the transition from RUN to STOP. After power up, the outputs are set to the default substitute value.

You configure the behavior of the outputs in Device Configuration. Select the individual devices and use the "Properties" pane to configure the outputs for each device. When the CPU changes from RUN to STOP, the CPU retains the process image and writes the appropriate values for both the digital and analog outputs, based upon the configuration.

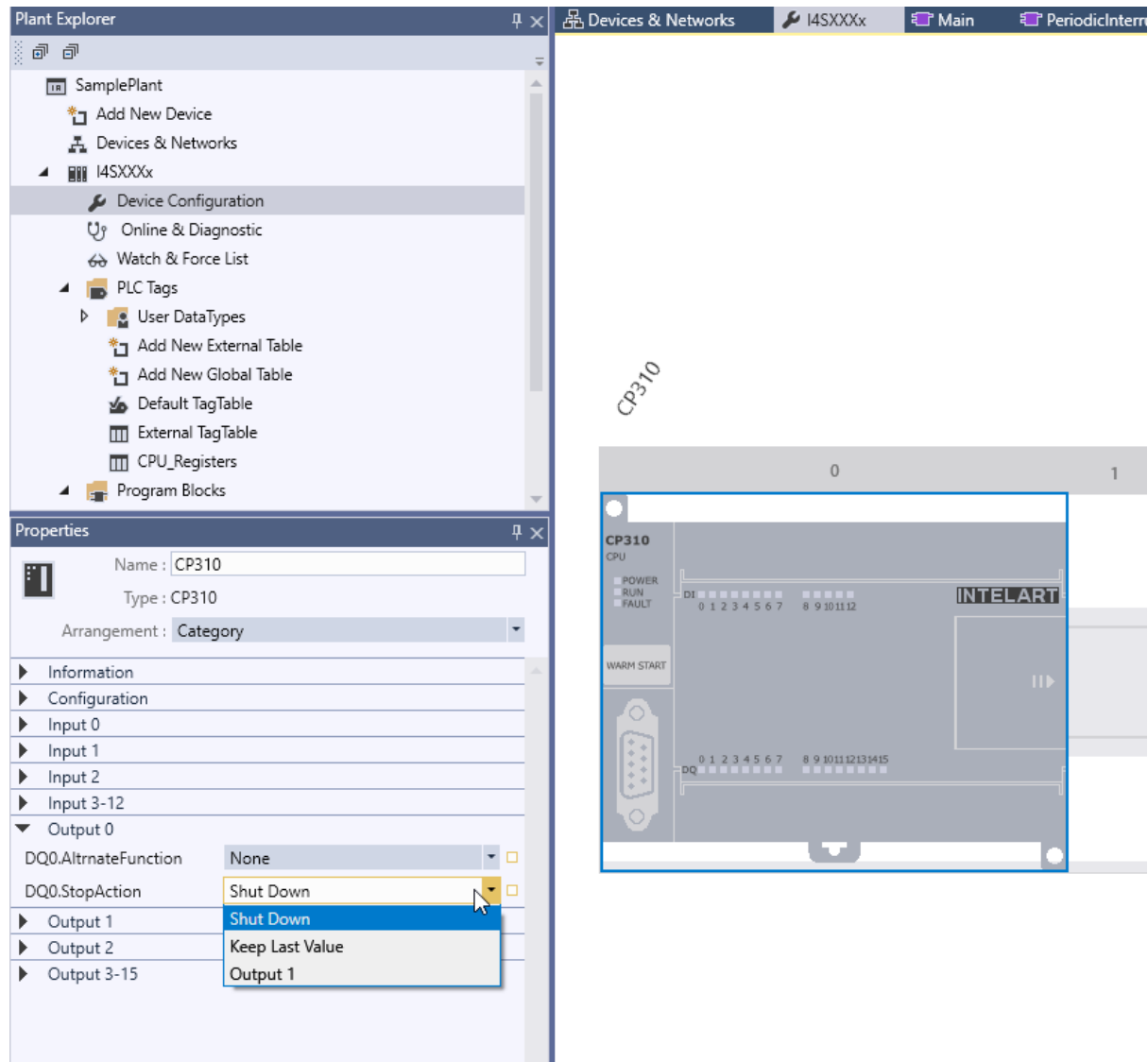


Figure 4-3 Choosing Stop Action for an output channel

2. Data storage, memory areas, I/O and addressing

2.1 Accessing the data of the I4 PLC

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

For a better understanding of how the CPU structures and addresses the memory areas, the following paragraphs explain the “absolute” addressing that is referenced by the PLC tags.

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

Global memory: The CPU provides a variety of specialized memory areas, including inputs (I), outputs (Q), reference memory (G) and bit memory (M). This memory is accessible by all code blocks without restriction

PLC tag table: You can enter symbolic names in the Intelart Studio PLC tag table for specific memory locations. These tags are global to all program blocks and allow programming with names that are meaningful for your application

Temp memory: Whenever a code block is called, the operating system of the CPU allocates the temporary, or local, memory (L) to be used during the execution of the block. When the execution of the code block finishes, the CPU reallocates the local memory for the execution of other code blocks.

Static Memory: Whenever a code block is called, the operating system of the CPU allocates the static local, memory (N) to be used during the execution of the block. But when the execution of the code block finishes, the CPU does not reallocate the static memory for the execution of other code blocks so its value will be fixed until the next execution of that code block.

Each different memory location has a unique address. Your user program uses these addresses to access the information in the memory location. References to the input (I) or output (Q) memory areas, such as I0.2.4 or Q2.1.7, access the process image.

Table 4-1 Memory areas

Memory Area	Description	Force	Retentive
I: Process Image Input	Copied from physical inputs at the beginning of the scan cycle	No	No
Q: Process Image output	Copied to physical outputs at the end of the scan cycle	Yes	No
M: Bit Memory	Control and data memory	Yes	No
L: Temp Memory	Temporary data for a block local to that block	No	No
G: Reference Memory	Control and data memory	Yes	Yes
S: Special Memory	CPU monitoring and control registers	No	No

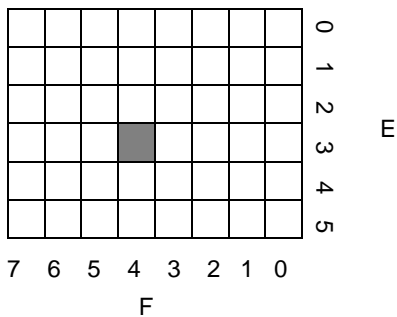
Each different memory location has a unique address. Your user program uses these addresses to access the information in the memory location. The absolute address consists of the following elements:

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

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

M 3 . 4

A B C D



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

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

2.1.1 Accessing the data in the memory areas of the CPU

Intelart Studio facilitates symbolic programming. Typically, tags are created either in PLC tags or in the interface at the top of an OB, FC, or FB. These tags include a name, data type, address, and comment. Additionally, in a reference tag table, a default value or retentive can be specified. You can use these tags when programming by entering the tag name at the instruction parameter. Optionally you can enter the absolute operand (memory area, size and offset) at the instruction parameter. The examples in the following sections show how to enter absolute operands. A % character must be inserted in front of the absolute operand in the program editor.

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

Table 4-2 Absolute addressing for I memory

Bit	I[physical module address].[byte address].[bit address]	I1.0.1
Byte, Word, or Double Word	I[physical module address].[size][starting byte address]	IB0.4, IW2.5, or ID18.12

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

Table 4-3 Absolute addressing for Q memory

Bit	Q[physical module address].[byte address].[bit address]	Q0.1.1
Byte, Word, or Double word	Q[physical module address].[size][starting byte address]	QB4.5, QW1.10, QD0.40

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

Bit	M[byte address].[bit address]	M18.7
Byte, Word, or Double word	M[size][starting byte address]	MB5, MW11, MD90

Temp (temporary memory): The CPU allocates the temp memory on an as-needed basis. The CPU allocates the temp memory for the code block at the time when the code block is called (for an FC or FB). The allocation of temp memory for a code block might reuse the same temp memory locations previously used by a different FC or FB. The CPU can initialize the temp memory at the time of allocation and therefore the temp memory can start by a specified value.

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

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

You access temp memory by symbolic addressing only.

Static (fixed memory): The CPU allocates the static memory on an as-needed basis for OBs and on creation for FBs. The CPU allocates the static memory for the code block at the time when the code block is called (for an OB). The allocation of static memory for a code block might reuse a temp memory location previously used by a different FC or FB. The CPU can initialize the static memory at the time of allocation and therefore the static memory can start by a specified value. The static memory does not reallocate after allocation by CPU. So its value always is fixed until a change by user program.

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

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

You access static memory by symbolic addressing only.

G (reference memory area): Use the reference memory area (G memory) for both control relays and data to store the intermediate status of an operation or other control information. Also Use the G memory for storing various types of data, including intermediate status of an operation or other control information parameters for FBs, and data structures required for many instructions such as timers and counters. You access reference memory by symbolic addressing only. You can mark a reference tag as "Retain" so its value will be retentive after CPU power off.

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

When you add a CPU and I/O modules to your configuration screen, I and Q addresses are automatically assigned. There is a tool for mapping I and Q addresses to a symbolic tags in an external tag table. See section 25.8 for more info.

Digital inputs and outputs are assigned in groups of 8 points (1 byte) and 16 points (1 word), whether the module uses all the points or not.

Analog inputs and outputs are assigned separately in real values (4 bytes)

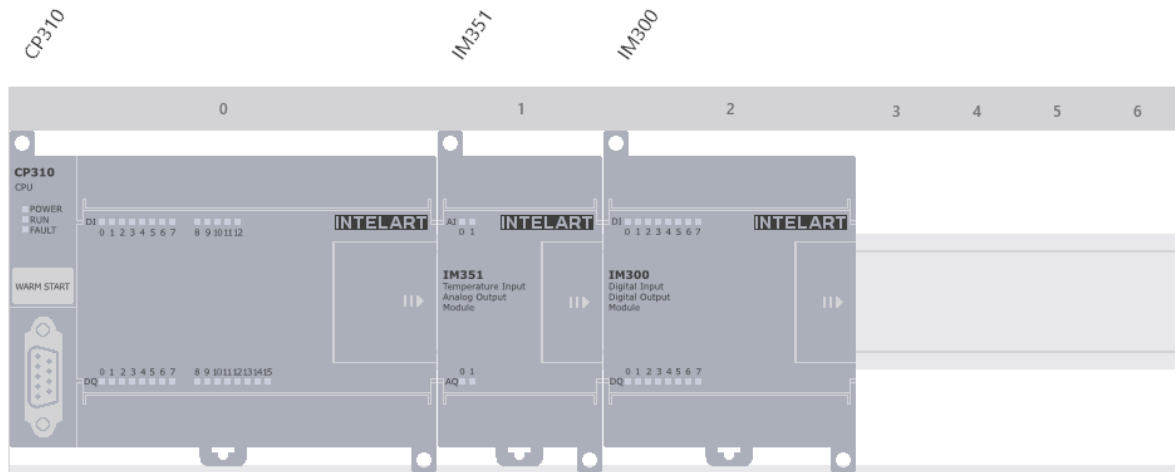


Figure 4-4 An example of a CPU CP310 with two expansion modules

3. Processing of analog values

Analog expansion modules provide input signals or expect output values that represent either a voltage range or a current range. These ranges are $\pm 10\text{V}$, $0 - 10\text{V}$, or $0 - 24\text{mA}$ or other special analog signals. The values returned by the modules are float (REAL) values having the exact value analog value. There is no need for conversion any word or integer to calculate real analog value. Anything outside the allowed range (overflow or underflow) will have a specific value. For each analog expansion module see its technical data for more info.

In your control program, you probably need to use these values in engineering units, for example to represent a volume, temperature, weight or other quantitative value. To do this for an analog input, you must normalize and scale the analog value to the minimum and maximum values of the engineering units that it represents. For values that are in engineering units that you need to convert to an analog output value, you normalize and scale value in engineering units to a value within $\pm 10\text{V}$, $0 - 10\text{V}$, or $0 - 24\text{mA}$, depending on the range of the analog module. Intelart Studio provides the "SCP_NORM" instruction in order to normalize and scale a numeric range to another range.

 TIP

When you use Temperature expansion modules you don't need to any conversion. The real tag values indicate the actual temperature by the module configuration. Just use its tag values in your program.
--

 TIP

In case you use Loadcell expansion modules you can use "WEIGH" instruction in order to execute a complete weighing system for calibrating and calculating actual weight or force value.

4. Data types

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

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

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

All data types are available in the PLC tags editor and the block Interface editors based on their scope. You can also enter a constant value for many of the input parameters.

- **Bit and Bit sequences:** Bool (Boolean or bit value), Byte (8-bit byte value), Word (16-bit value), DWord (32-bit double-word value), LWord (64-bit long-word value)
- **Integer:** USInt (unsigned 8-bit integer), SInt (signed 8-bit integer), UInt (unsigned 16-bit integer), Int (signed 16-bit integer), UDInt (unsigned 32-bit integer), DInt (signed 32-bit integer), ULInt (unsigned 64-bit integer), LInt (signed 64-bit integer)
- **Floating-point Real:** Real (32-bit Real or floating-point value), LReal (64-bit Real or floating-point value)
- **Time and Date:** Time (32-bit IEC time value), Date (32-bit IEC date value), TOD (32-bit IEC time-off-day value), DT (64-bit IEC date-and-time value)
- **Character and String:** Char (8-bit single character), String (64 byte-length string)
- **Array**
- **Data structure:** Struct
- **PLC Data type**
- **Pointers:** Any, Variant

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

Table 4-4 Size and range of the BCD format

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

4.1 Bool, Byte, Word, DWord and LWord data types

Table 4-5 Bit and bit sequence data types

Data type	Bit size	Number type	Number range	Constant examples	Address examples
Bool	1	Boolean	FALSE or TRUE	True, False	%I1.1.0, %Q1.0.1, %M50.7, Struct1.Tag2.3 TagName
		Binary	0 or 1		
		Octal	8#0 or 8#1		
		Hexadecimal	16#0 or 16#1		
Byte	8	Binary	2#0 to 2#11111111	2#00001111	%IB0.2, MB10, FB1.Tag4, TagName
		Unsigned integer	0 to 255	18	
		Octal	8#0 to 8#377	8#17	
		Hexadecimal	16#0 to 16#FF	16#F	
Word	16	Binary	2#0 to 2#1111111111111111	2#1111000011110000	%MW10, FB1.Tag2, TagName
		Unsigned integer	0 to 65535	18370	
		Octal	8#0 to 8#177777	8#170360	
		Hexadecimal	16#0 to 16#FFFF	16#F0F0	
DWord	32	Binary	2#0 to 2#11111111111111111111111111111111	2#1111000011111111 100001111	%MD10, FB1.Tag8, TagName
		Unsigned integer	0 to 4294967295	15793935	
		Octal	8#0 to 8#377777777777	8#74177417	
		Hexadecimal	16#0 to 16#FFFFFFFF	16#F0FF0F	
LWord	64	Binary	2#0 to 2#11111111111111111111111111111111 11111111111111111111111111111111 11111111111111111111111111111111	2#1111000011111111 11111111111111111111111111111111 111111100111111111 1111111111111100	%M1024.0, Struct3.Lword 2, TagName
		Unsigned integer	18446744073709551615	17446744063709551614	
		Octal	8#0 to 8#17777777777777777777777777777777	8#176777748775777 7727777	
		Hexadecimal	16#0 to 16#FFFFFFFFFFFFFFFF	16#FABFFF12FFFF 37F	

Table 4-6 Bit and bit sequence default value

Data type	Default value
Bool	False
Byte, Word, DWord, LWord	16#0

4.2 Integer data types

Table 4-7 Integer data types (U= unsigned, S= short, D= double, L= Long)

Data type	Bit size	Number range	Constant Examples	Address examples
USInt	8	0 to 255	78, 2#01001110	%MB0, FB1.B4, TagName
SInt	8	-128 to 127	+50, 16#50	
UInt	16	0 to 65,535	65295, 0	%MW2, FB1.Tag2, TagName
Int	16	-32,768 to 32,767	30000, +30000	
UDInt	32	0 to 4,294,967,295	4042322160	%MD6, FB1.DB8, TagName
DInt	32	-2,147,483,648 to 2,147,483,647	-2131754992	
ULInt	64	0 to 18446744073709551615	4294977295	%M20.0, FB1.Tag2, TagName
LInt	64	-9223372036854775808 to 9223372036854775807	-2347483648	

Table 4-8 Integer default values

Data type	Default value
USInt, SInt, UInt, Int, UDInt, DInt, ULInt, LInt	0

4.3 Floating-point real data types

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

Table 4-9 Floating-point real data types (L=Long)

Data type	Bit size	Number range	Constant Examples	Address examples
Real	32	-3.402823e+38 to -1.175 495e-38, ±0, +1.175 495e-38 to +3.402823e+38	123.456, -3.4, 1.0e-5	%MD0, FB1.R4, TagName
LReal	64	-1.7976931348623158e+308 to -2.2250738585072014e-308, ±0, +2.2250738585072014e-308 to +1.7976931348623158e+308	12345.123456789e40, 1.2E+40	%M12.0, FB1.R4, TagName

Table 4-10 Floating-point real default values

Data type	Default value
Real, LReal	0.0

TIP

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

4.4 Time and Date data types

Table 4-11 Time and date data types

Data type	Bit size	Range	Constant Examples	Address examples
Time	32	T#-596h31m23s648ms to T#-596h31m23s647ms	T#5m30s T#332h15m30s45ms	%MD0, FB1.T4, TagName %M12.0, FB1.R4, TagName
Date		D#1970-01-01 to D#2106-02-07	D#1992-02-07	
TimeOfDay		TOD#00:00:00 to TOD#23:59:59	TOD#10:20:30	
DateTime		DT#1970-01-01-00:00:00 to DT#2106-02-07-23:59:59	D#1992-02-07-03:28:11	

Table 4-12 Time and date default values

Data type	Default value
Time	T#0ms
Date	D#1970-01-01
TimeOfDay	TOD#00:00:00
DateTime	DT#1970-01-01-00:00:00

4.4.1 Time

TIME data is stored as a signed double integer interpreted as milliseconds. The editor format can use information for hours (h), minutes (m), seconds (s) and milliseconds (ms). It is not necessary to specify all units of time. For

example, T#5h10s is valid.

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

4.4.2 Date

DATE data is stored as an unsigned integer value which is interpreted as the number of seconds added to the base date 01/01/1970, to obtain the specified date. The editor format must specify a year, month and day.

4.4.3 TOD

TOD (Time of Day) data is stored as an unsigned double integer which is interpreted as the number of seconds since midnight for the specified time of day (Midnight = 0 s). The hour (24hr/day), minute, and second must be specified.

4.4.4 DT

DT (Date and Time) data is stored as an unsigned integer value which is interpreted as the number of seconds added to the base date time 01/01/1970 00:00:00. The editor format must specify a year, month, day, hour, minute and second.

4.5 Character and String data types

Table 4-13 Character and String data types

Data type	Bit size	Range	Constant Examples	Address examples
Char	8	ASCII character codes: 16#00 to 16#FF	'A', 't', '@'	%MB9, FB1.R4, TagName
String	64*8 (64 bytes)	64 bytes string	"ABC"	%M10.0, FB1.R4, TagName

Table 4-14 Character and String default values

Data type	Default value
Char	' '
String	""

4.5.1 Char

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

4.5.2 String

The CPU supports the String data type for storing a sequence of single-byte characters. The String type provides 64 bytes for storing the characters.

You can use literal strings (constants) for instruction parameters of type IN using double quotes. For example, "ABC" is a three-character string that could be used as input for parameter IN of the LEN instruction. You can also create string variables by selecting data type "String" in the block interface editors for OB, FC, FB, and other PLC tags editor.

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

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

Table 4-15 Valid ASCII control characters

Control characters	ASCII Hex value	Control function	Examples
\$\$	24	Dollar sign	"100\$\$", "100\$24"
\$'	27	Single quote	"\$'Text\$'", "\$27Text\$27"

\$N or \$n	0A	Line break	“\$NText”, “\$0A\$0DText”
\$R or \$r	0D	Carriage return (CR)	“\$RText”, “\$0DText”
\$T or \$t	09	Tab	“\$TText”, “\$09Text”

4.6 Array data type

You can create an array that contains multiple elements of the same data type. Arrays can be created in the block interface editors for OB, FC, FB, and other PLC tags editor.

To create an array from the block interface editor, name the array and choose data type type[length], then edit "length" and "type" as follows:

- Type: one of the data types, such as BOOL, SINT, UDINT
- Length: the length for your array

Table 4-16 ARRAY data type rules

Data type	Rules		
Array	<data type>[length1,length2,...] <ul style="list-style-type: none"> • All array elements must be the same data type. • The index must be greater than or equal to 0. • Arrays can have one to six dimensions. • Multi-dimensional index declarations are separated by comma characters. • Nested arrays, or arrays of arrays, are not allowed. • The memory size of an array = (size of one element * total number of elements in array) 		
	Array index	Valid index data types	Array index rules
	Constant or variable	USInt, SInt, UInt, Int, UDInt, DInt	<ul style="list-style-type: none"> • Value limits: 0 to 32767 • Valid: Mixed constants and variables • Valid: Constant expressions • Not valid: Variable expressions

Example: array declarations	Real[20] Int[11] String[2,3]	One dimension, 20 elements One dimension, 11 elements Two dimension, 6 elements
------------------------------------	------------------------------------	---

Example: array addresses	ARRAY1[0] ARRAY2[1,2] ARRAY3[i,j]	ARRAY1 element 0 ARRAY2 element [1,2] If i =3 and j=4, then ARRAY3 element[3, 4] is addressed
---------------------------------	---	---

4.7 Data structure data type

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

Arrays and structures can also be assembled into a larger structure. For example, you can create a structure of structures that contain arrays.

A Struct variable begins at an aligned-byte address based on CPU structure (usually 4-byte aligned).

4.8 User data type

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

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

If a new user data type is created and its interfaced updated (by clicking on "Update Interface" toolbar button in user data type editor), then the new user type name will appear in the data type selector drop drop-lists in the tag tables editor and code block interface editor.

Potential uses of user data types:

- User data types can be used directly as a data type in a code block interface or in tag tables editor
- User data types can be used as a template for the creation of multiple global data blocks that use the same data structure. For example, a user data type could be a recipe for mixing colors. You can then assign this user data type to multiple tag editors. Each tag editor can then have the variables adjusted to create a specific color.

4.9 Pointer data types

The pointer data types (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.

4.9.1 "Any" pointer data type

The pointer data type ANY ("Any") Only is available for the input / output/ in-out variables of system-defined Program Organization Units (POUs).

The Any pointer does not occupy any space in memory

4.9.2 "Variant" pointer data type

The data type Variant is a pointer 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.

The following diagram shows the structure of all PLC data types:

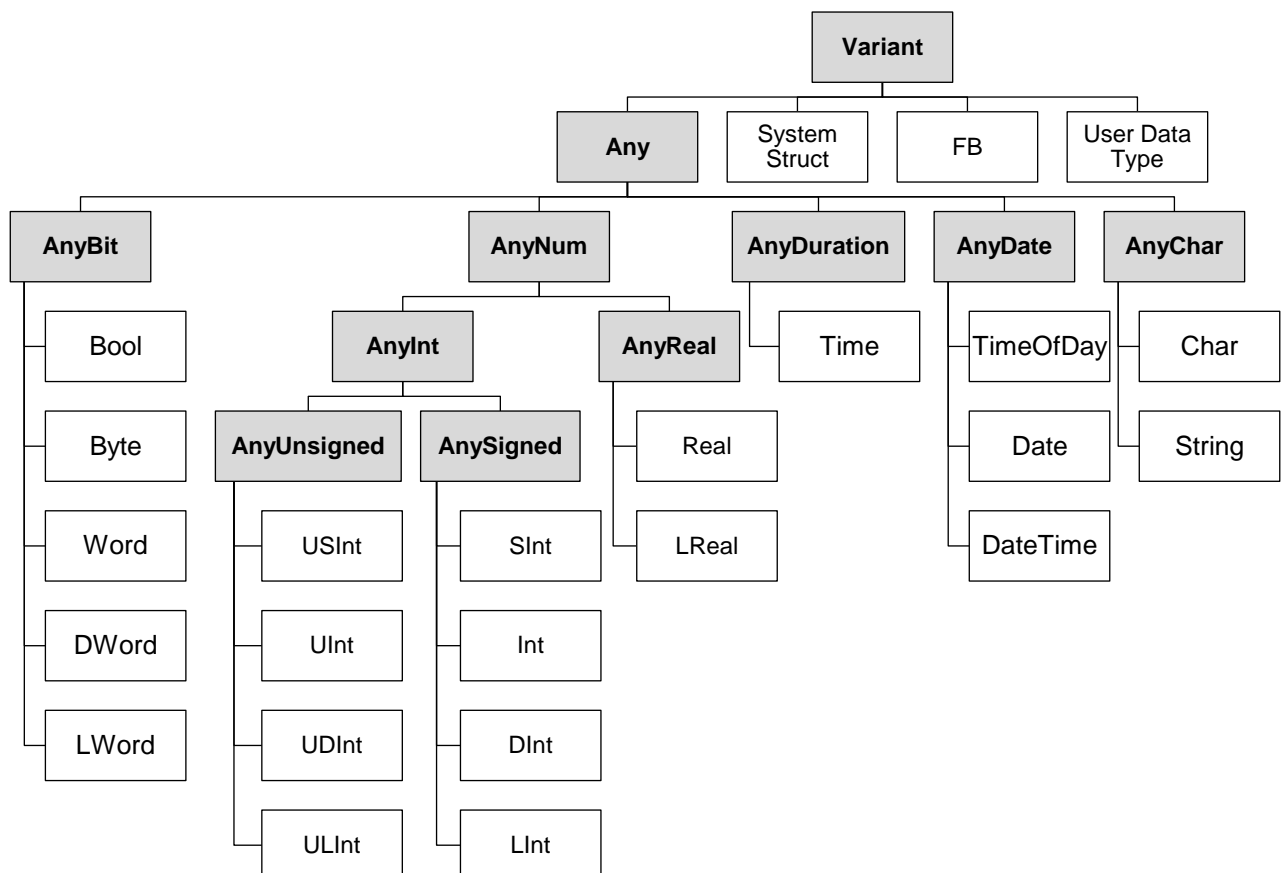
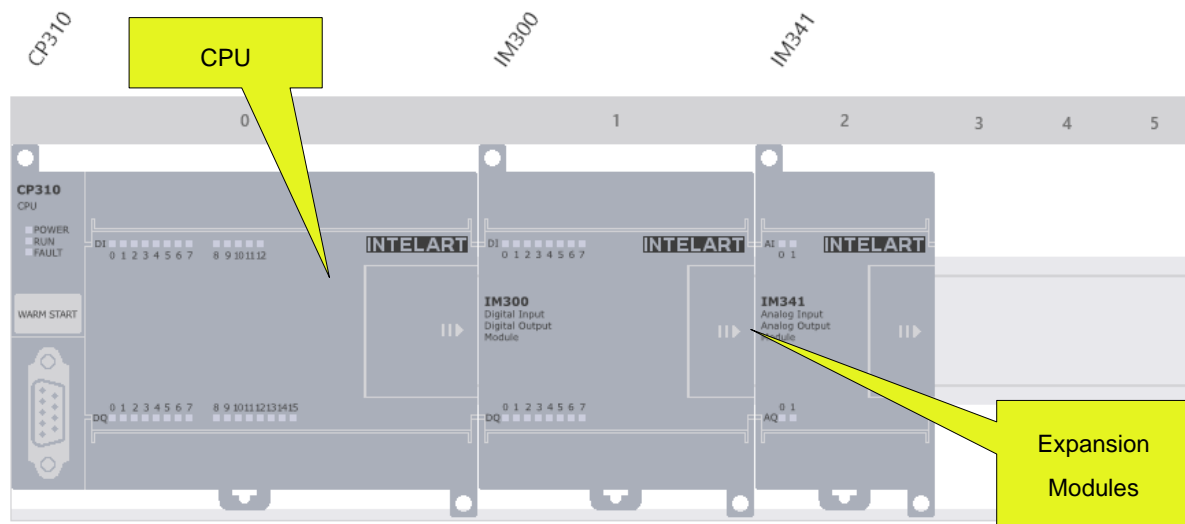


Figure 4-5 PLC data types diagram

5

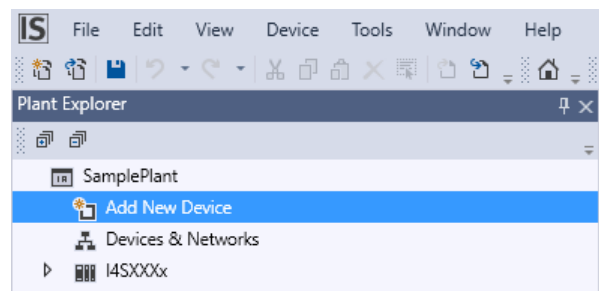
Device Configuration

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

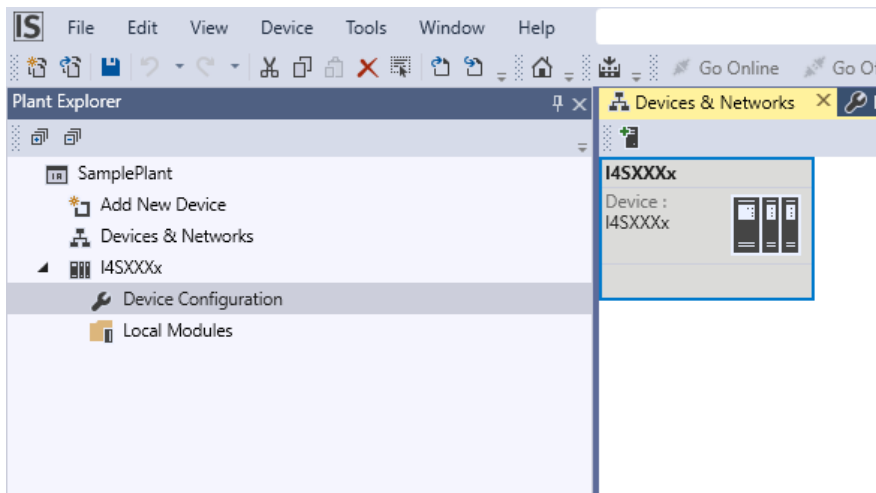
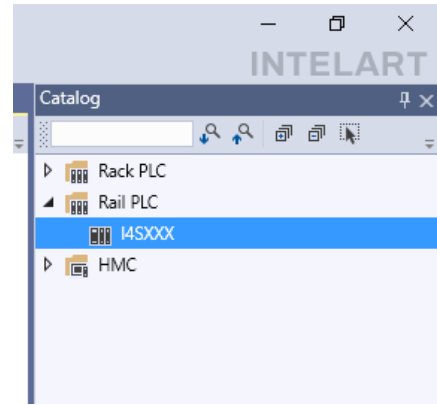


To create the device configuration, add a device to your project.

- 1- In the Plant Explorer pane double-click on "Add New Device" or "Devices & Networks".



- 2- In the Catalog pane double-click on a device or drag and drop it on the “Devices & Networks” editor.
- 3- A schematic of device will be added on the “Devices & Networks” editor. Also, in the Plant Explorer pane the device will be appeared.
- 4- Double click on the device schematic or on the “Device Configuration” in the Plant Explorer pane.
- 5- You can drag and drop a CPU or an expansion module on the slots of the rail or double-click on it in the Catalog pane.



1. Inserting a CPU

You create your device configuration by inserting a CPU into your project. Be sure you insert the correct model from the list. Select the CPU from the Catalog pane.

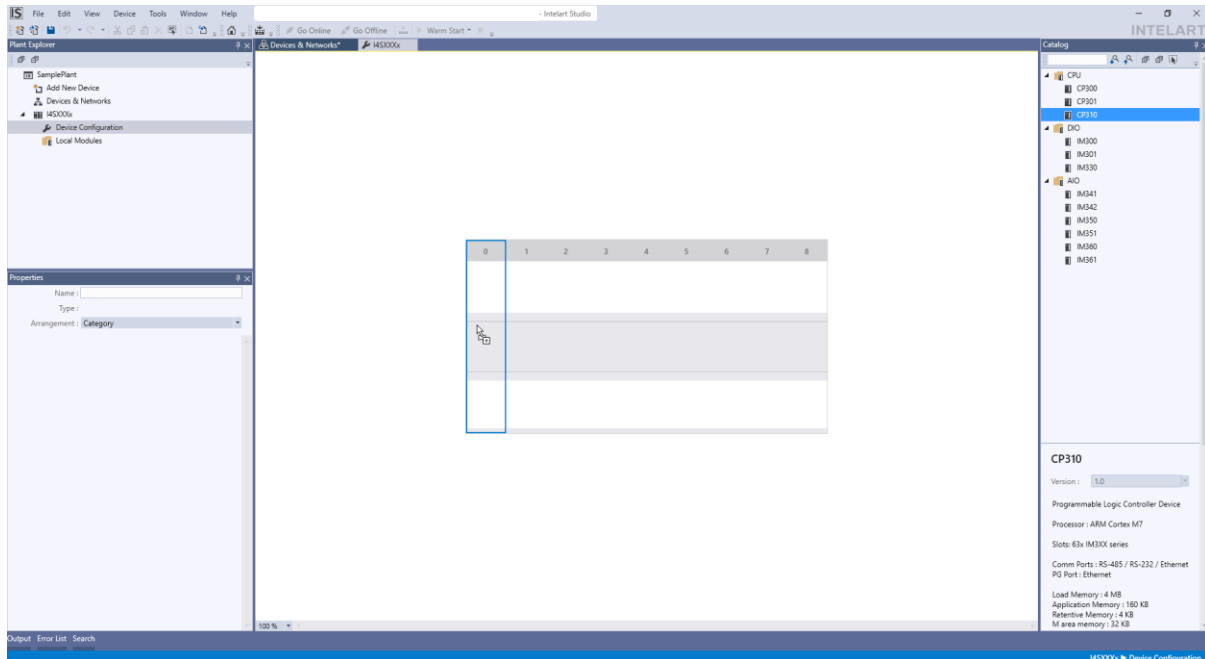
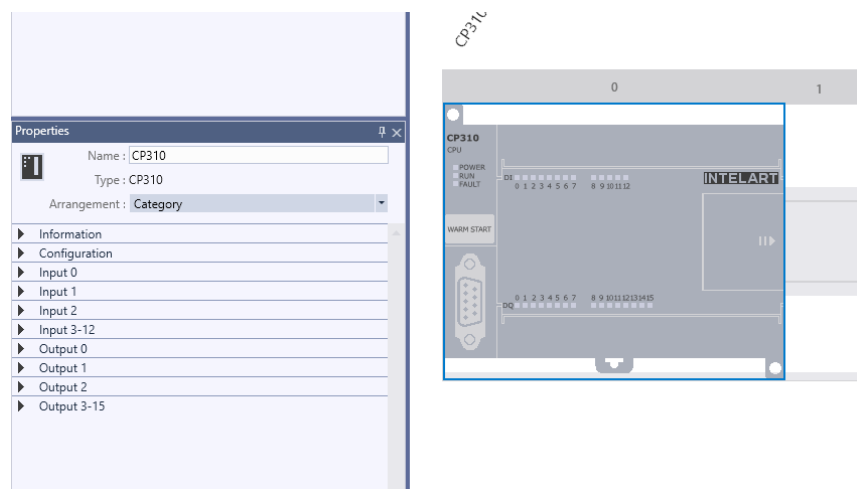


Figure 5-1 Inserting a CPU into the rail

Selecting the CPU in the Device view displays the CPU properties in the Properties pane.



NOTICE

The CPU has a pre-configured IP address 192.168.1.100. If your network does not support the default IP address You must manually assign an IP address for the CPU during the device configuration. If your CPU is connected by a serial protocol (such as USB), you may find the CPU COM port name in your device manager COM ports list.

If Intelart Studio does not find your I4 PLC, a Programmer Configuration dialog will be appeared in order to change the PG port parameters or search for available devices on a network.

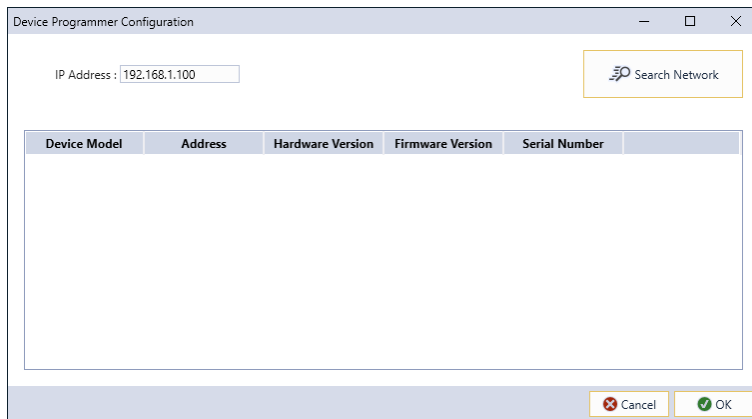


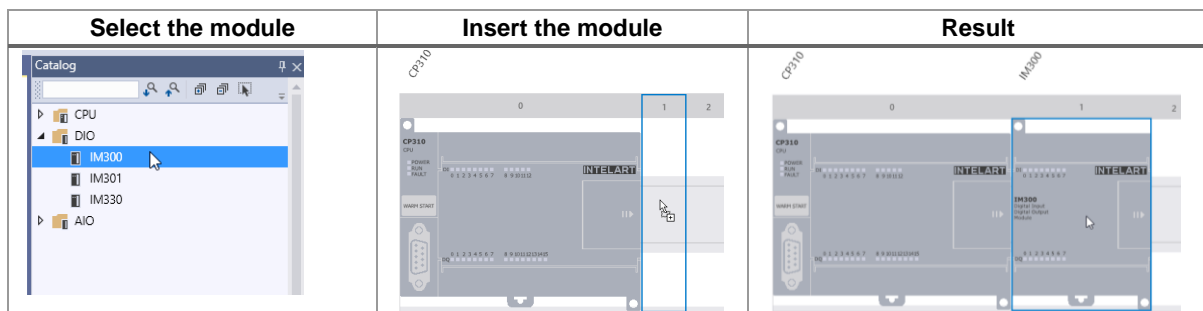
Figure 5-2 Programmer Configuration dialog

2. Adding modules to the configuration

Use the Catalog pane to add modules to the CPU:

Expansion module provides additional digital or analog I/O points. These modules are connected to the right side of the CPU. To insert a module into the device configuration, select the module in the Catalog pane 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



3. 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 pane in order to change configurations.

To configure input filter times, select an individual or grouped inputs. The default filter time for the digital inputs is 0 ms.

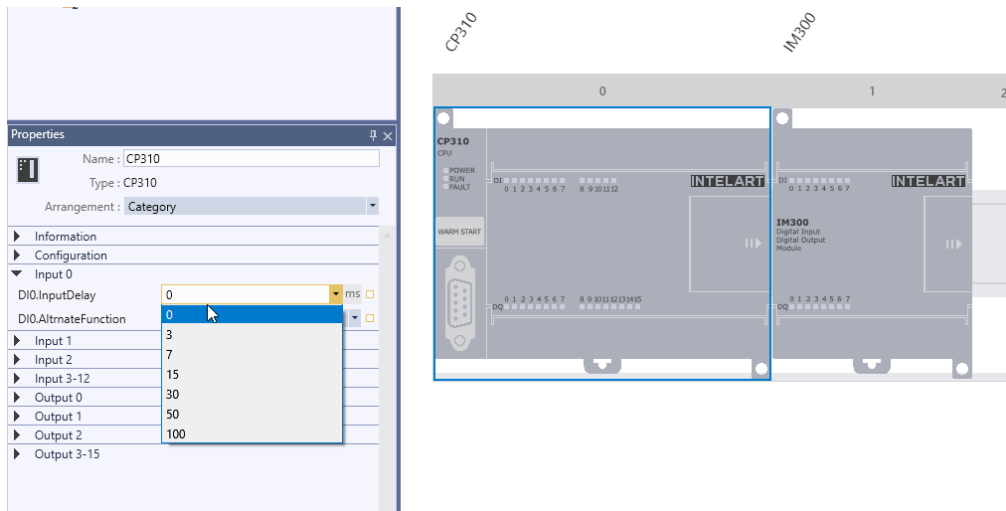


Figure 5-3 Changing CPU configuration

Property	Description
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 and pulse generators	Enables and configures the high-speed counters (HSC) and the pulse generators used for pulse-train operations (PTO), Frequency Out operations (FOO) and pulse-width modulation (PWM) When you configure the outputs of the CPU as pulse generators (for use with the PWM or motion control instructions), the corresponding output addresses (Q0.0, Q0.1,...) 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.
BindedSerialNumber	You can bind the CPU to a specific serial number so the program will be compiled only for that specific serial number. A CPU with a different serial number will not be programmed by the Intelart Studio.
Password	When a CPU is password protected, you must specify the password by this property in order to going online and programming the device.

4. 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 pane of the inspector window to configure the parameters for the module.

- Digital I/O: Some inputs can be configured for alternate functions based on their type and structure. See technical data in order to know the module configuration.
- 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 can configure ranges of analog inputs and outputs in engineering units on the Properties dialog.

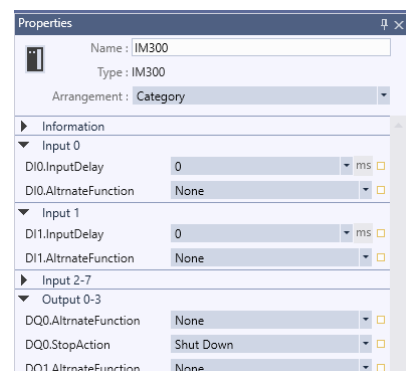


Figure 5-4 Configuring an expansion module

4.1 Assigning Internet Protocol (IP) addresses

4.1.1 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.

NOTICE

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.

NOTICE

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

Programming Device Adapter Card	Network Type	Internet Protocol (IP) Address	Subnet Mask
On-board adapter card	Connected to your plant LAN (and possibly the world-wide web)	<p>Network ID of your CPU and the programming device's on-board adapter card must be exactly the same.</p> <p>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.)</p>	<p>The subnet mask of your CPU and the on-board adapter card must be exactly the same.</p> <p>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.</p>
Ethernet-to-USB adapter card	Connected to an isolated network	<p>Network ID of your CPU and the programming device's Ethernet-to-USB adapter card must be exactly the same.</p> <p>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.)</p>	<p>The subnet mask of your CPU and the Ethernet-to-USB adapter card must be exactly the same.</p> <p>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.</p>

4.1.2 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- Go online to device and in the Plant Explorer pane, double-click on "Online & diagnostic".
- 2- In the status tab The MAC and other device information are displayed.

4.1.3 Modifying an IP address to a CPU online

You can assign an IP address to a network device online.

- 1- Go online to device and in the Plant Explorer pane, double-click on "Online & diagnostic".
- 2- In the Options tab click on "Load" button in order to load the current device configurations.
- 3- You can change either "IP Address", "subnet Mask" and "Gateway" fields.
- 4- By clicking on "Apply" button the current configuration will be transferred to de device.
- 5- In order to changes take effect, you must power off and the power on the device.

4.1.4 Configuring an IP address for a CPU in your project

IP address: Some CPUs 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 address 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 "IP" configuration entry. Intelart Studio displays the Ethernet address property in Properties pane, which associates the software project with the IP address of the CPU that will receive that project.

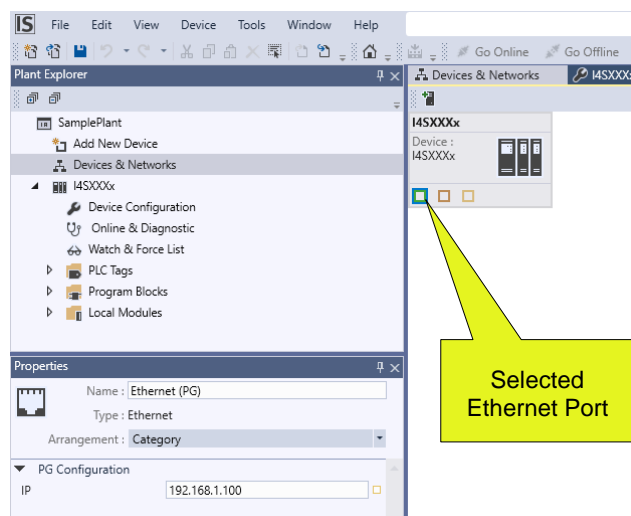


Figure 5-5 Assigning an IP for programming the CPU

 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 network may stop.

If the IP address of a CPU is changed to an IP address outside the subnet, the network will lose communication, and all data exchange will stop. User equipment may be configured to keep running under these conditions. Loss of 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.

6

Programming Concepts

In addition to the standard logical operations that a PLC can perform, seasoned PLC programmers are aware that, by taking advantages of some of the unique features and characteristics of a PLC, some very powerful operations can be performed. Some of these are operations that would be very difficult to realize in hardwired relay logic, but are relatively simple in PLC ladder programs. The reader should not concentrate on memorizing these concepts, but instead, learn how they work and how they can be best applied to solve programming problems

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	<p>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:</p> <ul style="list-style-type: none"> • 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 you require for access to your process. You can password-protect CPUs and program blocks from unauthorized access.
Specify the operator stations	<p>Based on the requirements of the functional specifications, create the following drawings of the operator stations:</p> <ul style="list-style-type: none"> • 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	<p>Based on the requirements of the functional specification, create configuration drawings of the control equipment:</p> <ul style="list-style-type: none"> • 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.

2. Structuring your user program

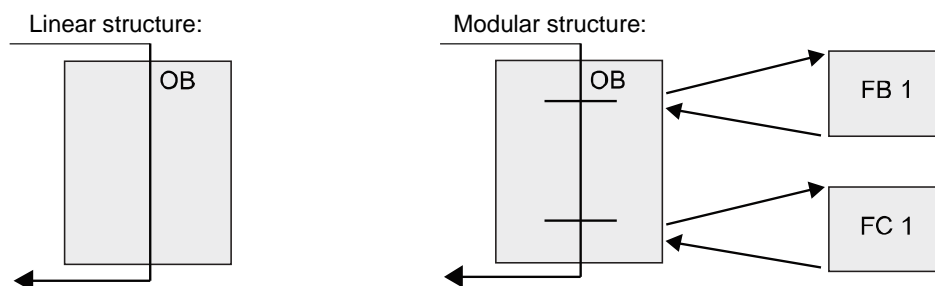
When you create a user program for the automation tasks, you insert the instructions for the program into code blocks:

- 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 (Default Main) 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 Main. 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 instance that stores the data for the specific call or instance of that FB. Changing the instance of FB 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 FBs 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 data. The calling block passes parameters to the FC. The output values from the FC must be written to a memory address or to a tag.

2.1 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 (Main Cyclic Program).
- A modular program calls specific code blocks that perform specific tasks. To create a modular structure, you divide the complex automation task into smaller subordinate tasks that correspond to the technological functions of the process. Each code block provides the program segment for each subordinate task. You structure your program by calling one of the code blocks from another block.



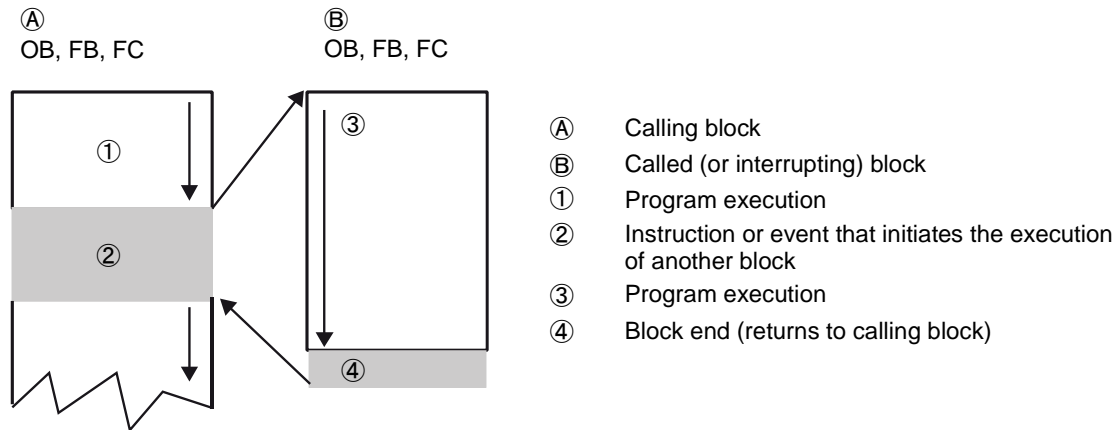
By creating generic code blocks that can be reused within the user program, you can simplify the design and implementation of the user program. Using generic code blocks has a number of benefits:

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

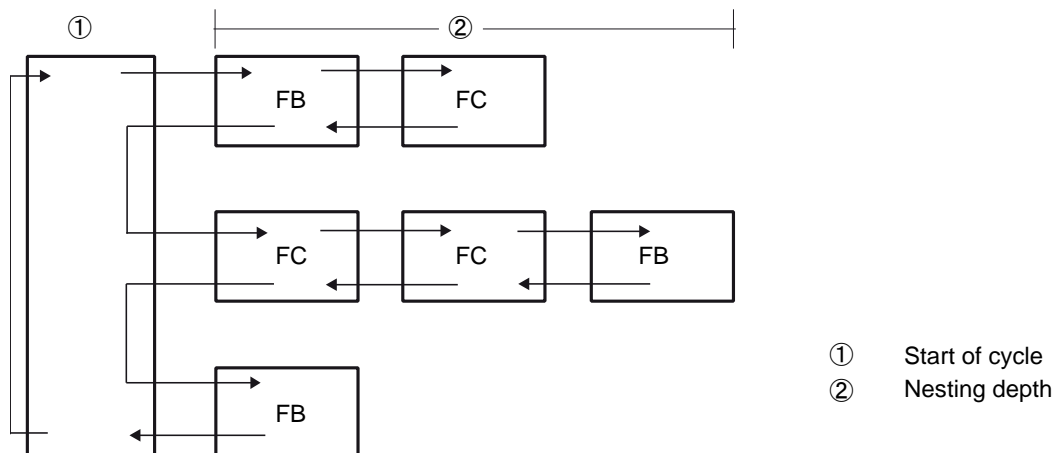
3. Using blocks to structure your program

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.



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.



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 cyclic program OB contains your main program. You can include only one cyclic program OB in your user program. During RUN mode, the cyclic program OB executes at the lowest priority level and can be interrupted by all other types of program processing. The startup OB does not interrupt the cyclic program OB because the CPU executes the startup OB before running the cyclic program.

After finishing the processing of the cyclic program OB, the CPU immediately executes the cyclic program OB 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 cyclic program 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 cyclic program OB.

Use the "Add New Program Block" dialog to create new OBs in your user program.

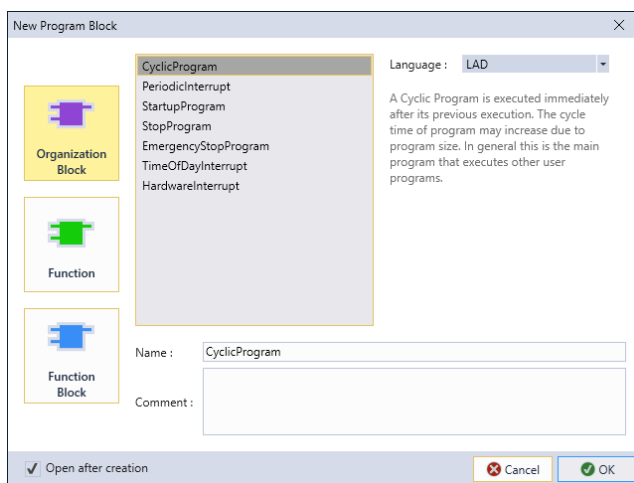


Figure 6-1 New Program Block dialog

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.

3.1.1 Creating an additional OB within a class of OB

You can create multiple OBs for your user program. Use the "Add New Program Block" dialog to create an OB. Enter the name for your OB and submit the new block.

3.1.2 Configuring the operation of an OB

You can modify the operational parameters for an OB. For example, you can configure the interval or priority parameter for a for a periodic interrupt OB.

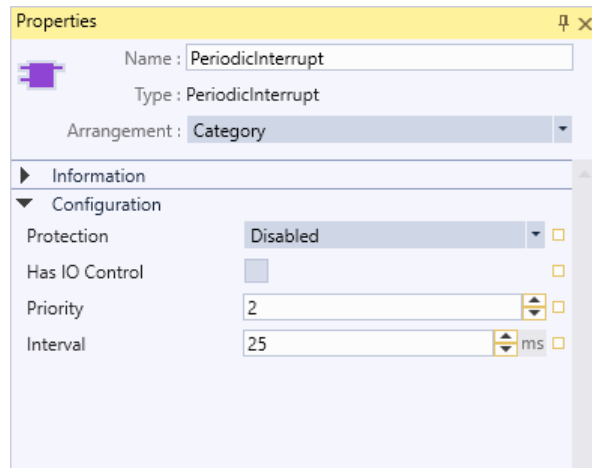


Figure 6-2 Periodic Interrupt properties

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. 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 or G memory.

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 named "FB instance", or "instance".

The instance provides a block of memory that is associated with that FB (or call) of the FB and stores data after the FB finishes. You can associate different instances with different calls of the FB. The instances 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 its instance. The CPU then executes the program code in that FB, and stores the block parameters and the static local data in the instance. When the execution of the FB finishes, the CPU returns to the code block that called the FB. The instance retains the values for that FB. These values are available to subsequent calls to the function block either in the same scan cycle or other scan cycles.

3.3.1 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 instances. When you call an FB, you also specify an instance that contains the block parameters and the static local data for that call or "instance" of the FB. The instance 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 instances for different calls of the FB.

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

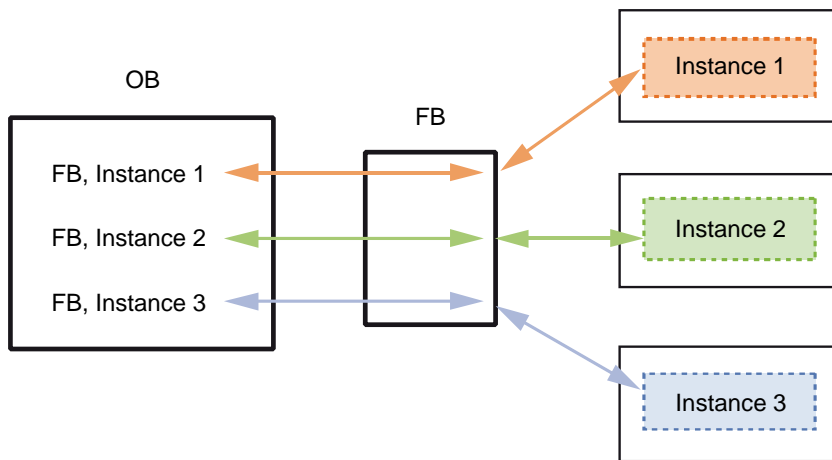
3.3.2 Assigning the start value in the instance

The instance 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 of FB. 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.

3.3.3 Using a single FB with multiple instances

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

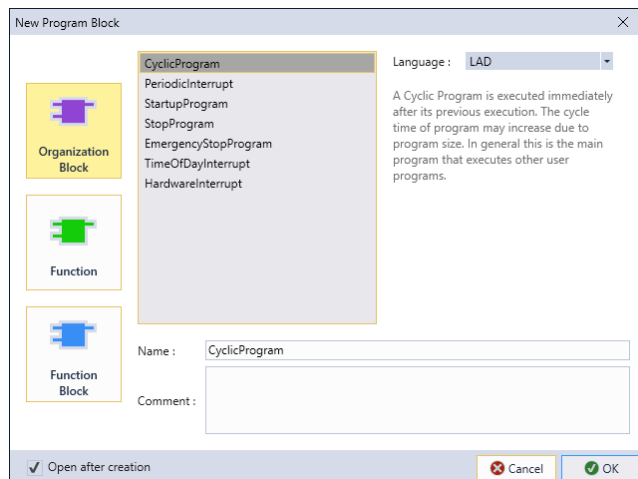


In this example, FB controls three separate devices, with Instance 1 storing the operational data for the first device, Instance 2 storing the operational data for the second device, and Instance 3 storing the operational data for the third device.

3.3.4 Creating reusable code blocks

Use the "Add New Program Block" dialog under "Program Blocks" in the Plant Explorer pane to create OBs, FBs and FCs.

When you create a code block, you select the programming language for the block.



TIP

When you make or change a user data type (UDT) or a function block (FB) structure, you must update its interface by clicking on the "Update Interface" button in the editor toolbar. The changes will not be taken effect until you update the interface of that data type.



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, TON or CTU).

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 cyclic program OB 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 cyclic program OB and an interrupt OB, your user program must also ensure that these values are modified or read consistently.

A communication request from an HMI device or another CPU can also interrupt execution of the cyclic program 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.

Ensure the data consistency for the buffers of data by avoiding any read or write operation to the buffers in both the cyclic program OB and an interrupt OB.

5. Programming language

Intelart Studio provides the following standard programming languages for I4 PLCs:

- LAD (ladder logic) is a graphical programming language. The representation is based on circuit diagrams.
- FBD (Function Block Diagram) is a programming language that is based on the graphical logic symbols used in Boolean algebra.

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.

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.

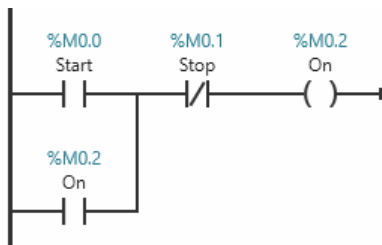


Figure 6-3 A sample ladder network

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

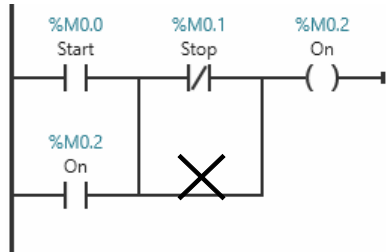
Intelart Studio does not limit the number of instructions (rows and columns) in a LAD network.

TIP

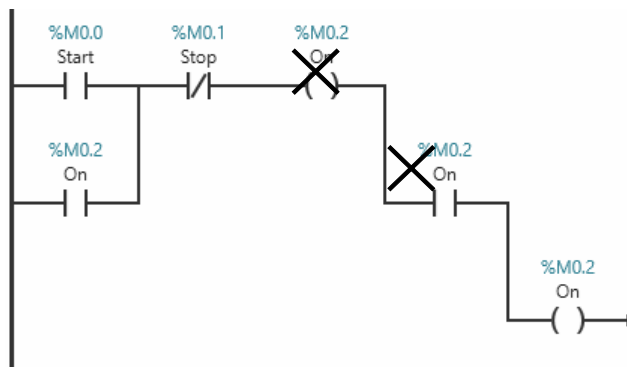
Every LAD network must terminate with a coil or a box instruction.

Consider the following rules when creating a LAD network:

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



- A ladder network must be flowed from left top to right bottom direction



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.

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

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

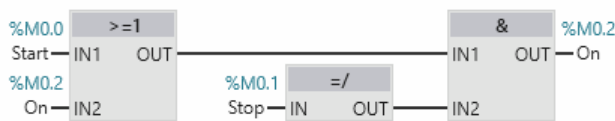


Figure 6-4 A sample function block diagram network

5.3 EN and ENO for LAD and FBD

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

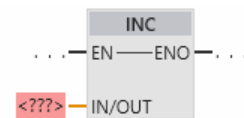
Certain instructions (such as the Math and the Move instructions) provide parameters for EN and ENO. These parameters relate to power flow in LAD or FBD and determine whether the instruction is executed during that scan.

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.

TIP

Some instructions have a short circuited EN-ENO. It means that the ENO is directly passes the EN value and is not dependent on the functionality of that instruction.



6. Protection

6.1 Access protection for the CPU

The CPU provides a security mechanism for restricting access to going online. When you configure the password for a CPU, you limit the communications with the Intelart Studio that cannot be accessed without entering a password.

The password is case-sensitive.

To configure the password, follow these steps:

- 1- In the "Online & Diagnostic", o to Options.
- 2- The Intelart Studio must be gone online and the CPU must be in STOP mode.
- 3- Click on load button in order to loading the current configuration of CPU.
- 4- The "Password" field never loads by the current password because of security reasons.
- 5- Specify the password by entering it in the "Password" field.
- 6- Click on "Apply" button to download the current configuration to the CPU.

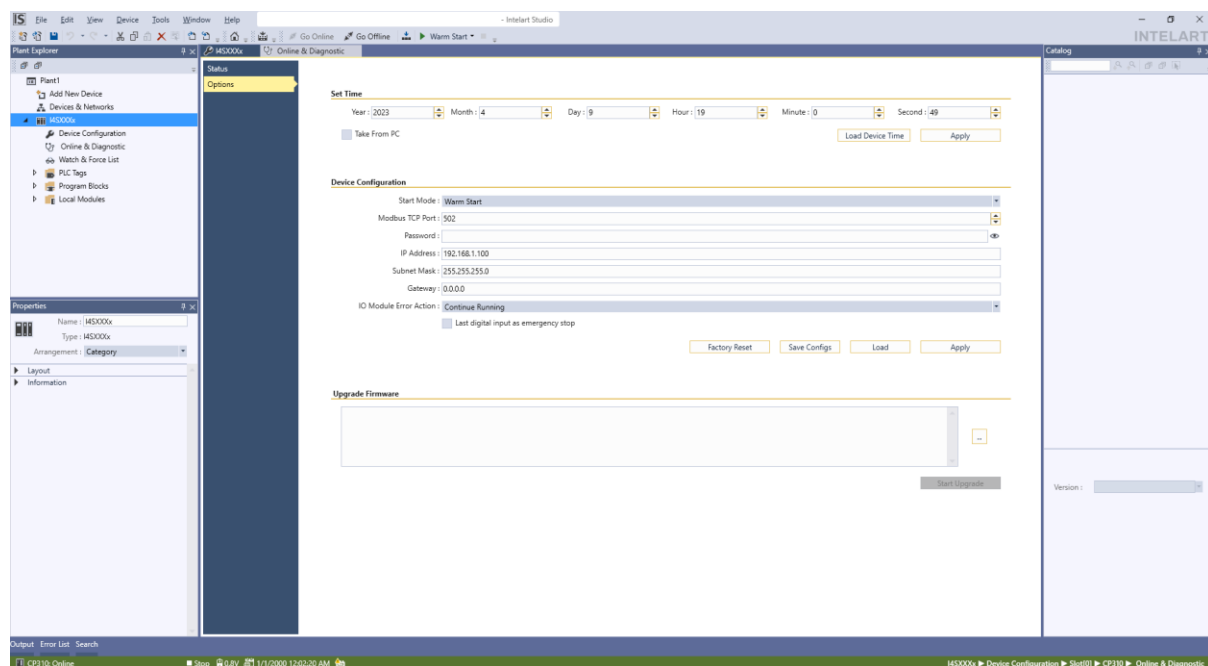


Figure 6-5 Configuration of a CPU

TIP

You can save current configuration in a ".iacfg" file in order to use in future or send to another person.

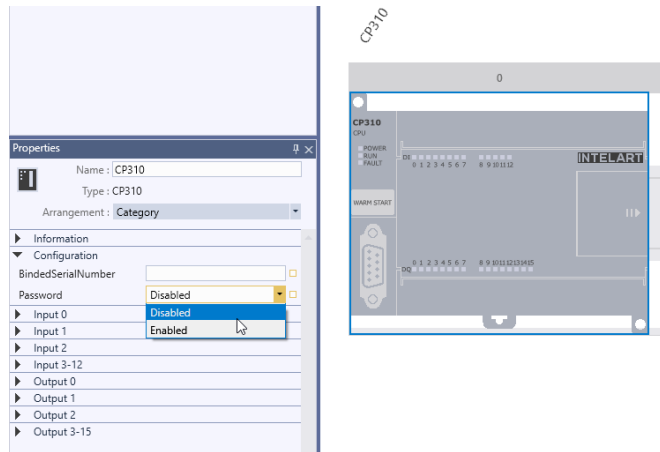
TIP

A restart is needed after downloading the configuration to the CPU in order to settings will be take effect. You can simply turn off the power and then turn it on after a few seconds.

6.1.1 Going online to a protected CPU

In order to going online to a protected CPU, follow these steps:

- 1- In the "Device configuration", select the CPU.
- 2- In the Properties pane, expand the "Configuration group".
- 3- Select the "Password" property to enable the protection and to enter a password.
- 4- Click on "OK" button in order to save the changes.
- 5- Click on "Go Online" button in main toolbar.



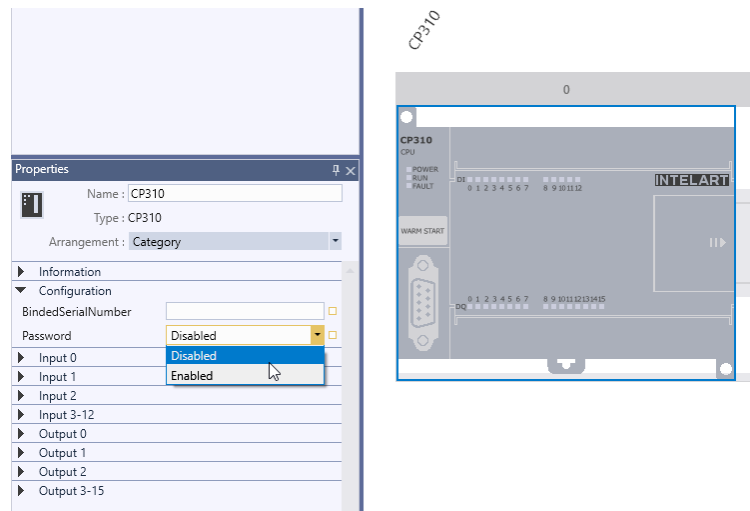
6.2 Program blocks protection

Program block protection allows you to prevent one or more code blocks (OB, FB or FC) in your program from unauthorized access. You create a password to limit access to the code block. The password-protection prevents unauthorized reading or modification of the code block. Without the password, you can read and edit Block tags editor, block code body, and block properties.

When you configure a block for "Enabled" protection the Intelart Studio asks you a password to protect the program block. After that the user must enter the password in order be able to view and edit the code body and tags list otherwise the code within the block will not be accessed. Also, the tags list will be read only and write protected.

Use the "Properties" pane of the code block to configure the protection for that block. After opening the code block, select "Protection" from Properties.

- 1- In the Properties for the code block, click the "Protection" drop down list to display the protection modes list.
- 2- Click on "Enabled" item and then enter the password in the password dialog.
- 3- Click on "OK" button in order to save changes.
- 4- Close the program block editor and try to reopen it. You will see a pop-up dialog asks you a password to proceed.
- 5- If you enter the correct password, all program block parts will be editable for you.
- 6- If you click on "Cancel" button, the program block editor will be opened in limited mode (described earlier).



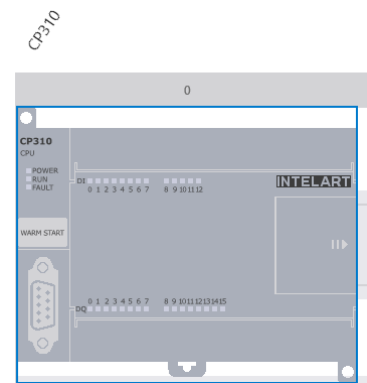
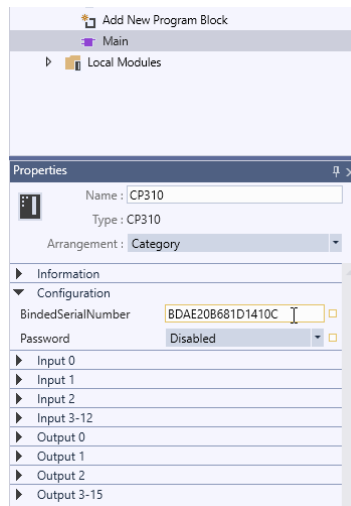
TIP

If you choose "permanent" mode for protection of a program block, it will be protected permanently and any user will not be able to edit the program block even by accessing the password.

6.3 Copy protection

An additional security feature allows you to bind the compiled project for use with a specific CPU. This feature is especially useful for protecting your intellectual property. When you bind a project to a specific device, you restrict the program and all its code blocks for use only with a specific 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 device.

- 1- In the "Device configuration", select the CPU.
- 2- In the Properties pane, expand the "Configuration group".
- 3- Click on the "BindedSerialNumber" property to enter a specific CPU serial number.
- 4- Now you can compile the project and send the compiler binary output file for any person. The compiled project will be downloaded on a CPU with the specified serial number in the "BindedSerialNumber" property.



TIP

The serial number is not case sensitive.

TIP

You can access the compiled binary output file in the "output" folder beside the ".iapln" plant file.

6.4 Downloading a compiler binary output file

You can download a CPU program without needing the source project. Follow these steps:

- 1- Click on "Compile" button in main toolbar (or press F7 key on keyboard).
- 2- Open the "output" folder beside the ".iapln" plant file. You will see a ".iabin" file (with the device name in the project).
- 3- This file contains all program data in order to download without needing to access the source project file.
- 4- In the "Device" menu, click on the "Download Pre-Compiled" menu item.
- 5- You will see a dialog asks for "Compiled Application" (".iabin" file) and "Device Configuration" (".iacfg" file) files selectable for download to the CPU.
- 6- Determine each part you need and click on "Download" button.



TIP

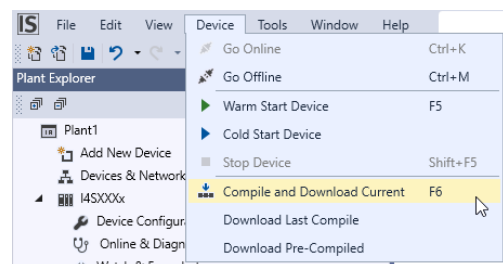
If a ".iabin" file is copy protected by binding to a specific serial number, you will be able to download that file only on the specified CPU but the configuration file always will be downloaded to the CPU.

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 other device configurations) in a nonvolatile memory (Load Memory).

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

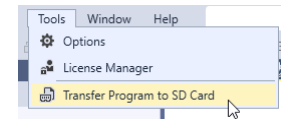
- "Device" menu: Click the "Current and Download " or "Download Last Compile" or "Download Pre-Compiled" menu item.



- **Toolbar:** Click the "Compile Current and Download" button or press F6 keyboard key.



- **"Tools" menu:** Click the "Transfer Program to SD Card". Some devices may not support this feature. For more info see the next section.



7.1 Transfer Program to SD Card

You can program a device by a SD card memory If the device supports programming by SD card. Follow these steps:

- 1- Insert the SD card into a memory card reader or any other memory card reader and connect the device to your programming device (PC).
- 2- Click on "Transfer Program to SD Card" menu item in "Tools" menu.
- 3- Open the "output" folder beside the ".iapIn" plant file. You will see a ".iabin" file (with the device name in the project).
- 4- This file contains all program data in order to download without needing to access the source project file.
- 5- You will see a dialog asks for "Compiled Application" (".iabin" file) and "Device Configuration" (".iacfg file) files selectable for download to the CPU.
- 6- Determine each part you need and click on "Download" button.
- 7- You can tell the CPU you want the program data files be removed after downloading program process by enable the "Delete data files after download" check box (default is checked).
- 8- Click on "Save Files" button and in the directory selection dialog choose the SD card memory root address.
- 9- Click Ok and wait for a moment in order to files be wrote on the memory then safely remove the SD card.
- 10- Insert the SD card in the SD card slot in the CPU when the device is turned off.
- 11- When you turn on the device, it will detect the files and will start the downloading process automatically. wait for the process to finish. Then remove the SD card.

TIP

You cannot download a Copy Protected program by SD card to CPU.

8. Uploading from the CPU

You cannot upload any part of program from CPU due to security reasons. You must take care of your project files and keep them in a safe place in order to future use.

9. Monitoring and testing the program

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 6-2 Monitoring and modifying data with Intelart Studio

Editor	Monitor	Modify	Force	Log
Watch & Force list	Yes	Yes	Yes	Yes
Program editor	Yes	Yes	No	No

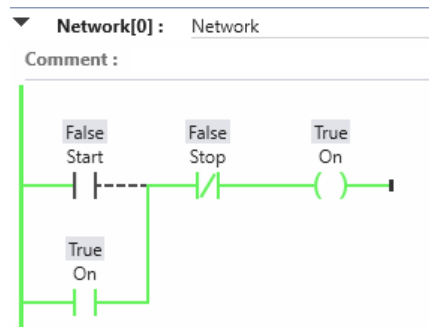


Figure 6-7 Monitoring with the LAD editor

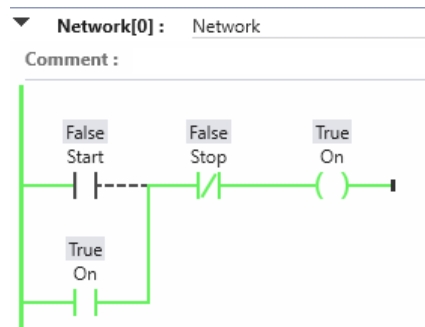


Figure 6-6 Modifying with the LAD editor

	Name	Data Type	Address	Display Format	Monitor Value	Modify Value	Captured Value	Select	Log	Comment	Tag Comment
1	Start	Bool	%M0.0	Default	False	False		<input checked="" type="checkbox"/>	<input type="checkbox"/>		
2	Stop	Bool	%M0.1	Default	False	False		<input checked="" type="checkbox"/>	<input type="checkbox"/>		
3	On	Bool	%M0.2	Default	True	False		<input checked="" type="checkbox"/>	<input type="checkbox"/>		

Figure 6-8 Monitoring, Modifying, Forcing and logging with a watch & Force List

9.2 Watch and force list

You use "Watch & Force List" for monitoring, modifying, Forcing and Logging the values of a user program being executed by the online CPU. With a Watch & Force List, 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, but also for the memory areas of the CPU, including the inputs and outputs (I and Q), special memory (S), bit memory (M), and reference memory (G).

With the Watch & Force List, you can modify or force the physical digital and analog outputs (Q) of a CPU. For example, you can assign specific values to the outputs when testing the wiring for the CPU.

TIP

You cannot force an input (or "I" address).

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" dialog displays the instances where a selected object is being used and the other objects using it. To display the cross-references, select the "Cross References" in context menu of that object.

TIP

By double-click on the Cross References dialog items you can navigate to that object.

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 and the relationships between 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. Intelart Studio 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 or FBs 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.

7

Basic Instructions

This chapter describes basic instructions based on IEC 61131-3. Also, some instructions have been added in addition to the original IEC instructions.

Before you start programming a CPU, you must know the following concepts. Some instructions may support none, some, or all of these concepts. All configurable instructions emerge a “configuration” group in the Properties pane when you are select them by mouse. To be able to use all these features, you need to know these features:

- Instruction Type:** you can change the operation logic of some instructions without needing to delete them and add new instructions of that group. You just can easily change the logic (or type of that instruction) by changing the `InstructionType` property.

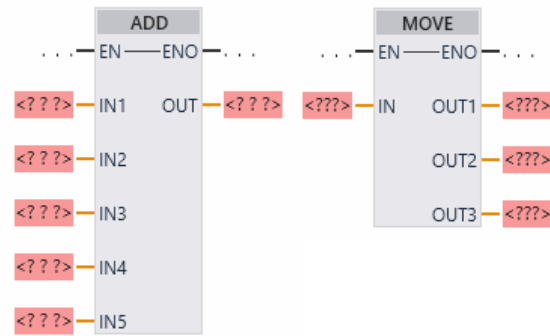
Example: you want to change a wired AND instruction to OR but you have set the arguments so deleting and adding a new instruction will be awkward. You can make the AND to OR by changing the `InstructionType` property or by double-click on its name in top of the instruction box and select another instruction type.

The screenshot illustrates the configuration of an AND instruction. At the top, a dropdown menu for 'InstructionType' is open, showing options: AND (selected), OR, and XOR. Below this, the 'OperationDataType' is set to AND and 'InputsCount' is 3. The main part of the image shows a ladder logic diagram with an AND instruction box. The box has three inputs: EN-1, IN1, and IN2. The IN1 and IN2 inputs are currently set to '<???'.

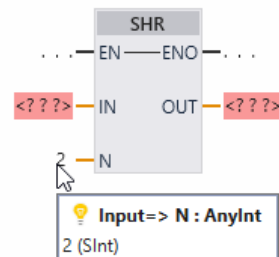
- Operation Type:** Some instructions can operate on multiple internal data types. You can select the most suitable data type for your instruction by changing the “`OperationDataType`” property in Properties pane. Example: you have inserted a ADD instruction and it is going to add real tags together. In this case you must change the “`OperationDataType`” property to “Real”.

The screenshot illustrates the configuration of an ADD instruction. At the top, a dropdown menu for 'OperationDataType' is open, showing options: Int (selected), SInt, Int, DInt, LInt, USInt, UInt, UDInt, ULInt, Real, and LReal. Below this, the 'InstructionType' is set to ADD and 'InputsCount' is 2. The main part of the image shows a dropdown menu for 'InstructionType' with options: ADD (selected), AND, OR, and XOR.

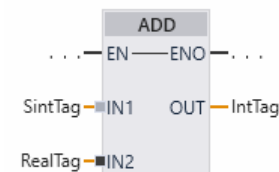
- Inputs/Outputs Count:** If an instruction supports variable inputs or outputs count, you can change them by the "InputsCount" or "OutputsCount" properties in properties pane. Example: you want to add 5 numbers together. Instead of using multiple ADD instruction, you can change "InputsCount" property to 5 so the inputs of the ADD instruction will be increased to 5.



- Implicit Casting:** In implicit casting, the conversion involves a smaller data type to the larger type size. For example, the SInt datatype implicitly cast into Int. The process of converting the lower data type to that of a higher data type is referred to as internal widening in the instruction operations. Example: You want to specify the number of shifting in the SHR instruction. As the N input is of type AnyInt, you can enter any integer with any size (such as USInt or UInt) or a constant. The conversion process will be automatically without error in the internal instruction operations.



- Explicit Casting:** Explicit type conversion, also called type casting, is a type conversion which is explicitly defined within a program (instead of being done automatically according to the operation of instruction or implicit type conversion). It is requested by the user in the program. Example: An ADD instruction with Int Operation Data Type can accept any number (AnyNum) on its arguments. If the argument assigned tag data type does not match the instruction type, an explicit casting will be occurred automatically before execution of ADD instruction internal operations.



⚠ WARNING

Sometimes an explicit casting can lead to runtime error in CPU. For example, if you try to cast an Int with a value 1000 to an SInt value, a runtime error will be occurred. Explicit casting with the probability of runtime error emerges as a bold square beside the instruction argument.

The diagram shows an ADD instruction block with EN and ENO terminals. It has two input terminals: IN1 with 'SintTag' next to it and IN2 with 'RealTag' next to it. There is one output terminal labeled OUT with 'IntTag' next to it. A mouse cursor points to the IN2 terminal, and a box labeled 'Explicit Casting' is positioned next to it.

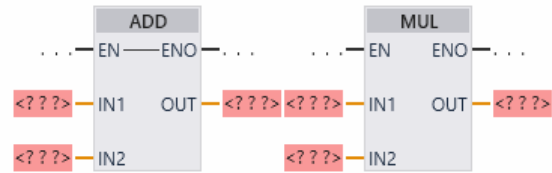
A yellow warning tooltip box with a warning icon and the text: 'Warning Explicit Casting. Runtime error is possible'.

- EN/ENO Arguments:** In LAD or FBD some instructions provide input EN and output ENO arguments. For these instructions, if you want to allow the execution of the instruction, set its EN to 1. If you want to suppress the execution of an instruction, set its EN to 0. If the block is executed correctly, output ENO follows input EN. If an error is detected while executing an instruction with internal error diagnostic, output ENO of that instruction is set to value 0. The values for all other outputs and in-out variables of the instruction are in principle undefined. That means that these variables can assume different values on different operation states.

💡 TIP

When to use ENO with instructions providing an internal error diagnostic?
 If you want to clear the execution of following blocks but only when the current block is executed without errors, then make the execution of the following blocks dependent on output ENO of the current block (for example by an appropriate access to output ENO).

- EN-ENO Connected:** You may see some instructions have a line between their EN and ENO arguments. It means the ENO follows the EN value regardless of instruction execution result. Example: ADD instruction is an EN-ENO connected instruction but MUL instruction is not.



- Wire and non-Wire Arguments:** In FBD you can connect multiple instructions by a wire together. You can determine whether an argument can be wired or not by checking the argument indicator. Space between character (such as “. . .” or “? ? ?”) instead of “. . .” or “????”) indicates that it can be wired to another argument. Example: OUT and IN arguments in SHR and SHL instructions accept wire but the N argument do not.



TIP

Wiring instructions in FBD:

- Press down left mouse button on an output argument and then drag the mouse while the left mouse button is hold down
- Move the mouse cursor over an input argument that you desire to make a wiring between them.
- when the wiring between the two arguments is eligible the mouse cursor turns to a green icon indicating you release the mouse button to make a wire connection between the arguments. Otherwise, the mouse cursor icon remains in a red status.

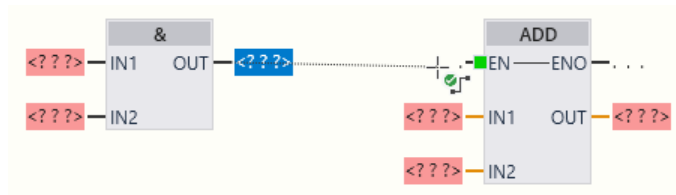


Figure 7-1 An eligible wiring

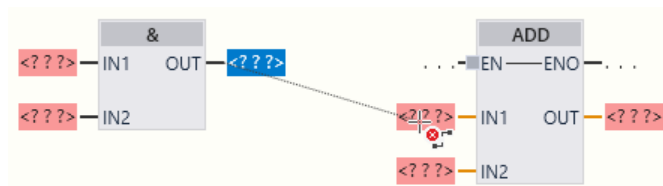


Figure 7-2 An unauthorized wiring

- Optional Argument:** Some instructions have one or more arguments that is not necessary to assign a tag (or constant). These arguments will be indicated by “. . .” or “. . .”. In case you do not assign a tag (or constant) to an optional argument, its default value will be determined.
- Limited/Unlimited Function Array Argument:** When you design a FC or FB, you can define arrays of unlimited size for some arguments. This feature makes your FCs and FBs not dependent on array size so you can design more general FCs or FBs. In order to define an unlimited array, you should define an array like a limited size array but place a “*” instead of a number for each dimension size.

Example:

	Name	DataType	DefaultValue	Comment
Input				
	UnlimitedBool	Bool[*]		
	Unlimited2DWord	Word[*,*]		
	<Add New Item>			

TIP
 You can define unlimited arrays for Input, Output and InOut in FCs and only for InOut in FBs.

- **Abstract Function Argument:** When you design a FC or FB, you can define an abstract data type instead a basic data type. For example, you can define an AnyInt instead of Int so you will be able to assign SInt, Int, DInt and LInt tags to that argument when you call your FC or FB.

Example:

Name	Data Type	DefaultValue	Comment
IntegerInput	AnyInt		
FloatInput	AnyReal		
<Add New Item>			

TIP
 You can define abstract data types for Input, Output and InOut in FCs and only for InOut in FBs.

- **Local/Global Argument Notation:** You can determine whether an assigned tag to an instruction argument is global or local by the prepended character “#” for locals. That means if a local tag assigns to an instruction argument, its name will be started by a “#” sign. Example: Assigned tag to IN1 is a global tag but the assigned tag to IN2 is a local tag.



TIP
 All I, Q, M and G tags are global elsewhere are local.

1. Bit logic

1.1 Bit logic contacts and coils

LAD and FBD are very effective for handling Boolean logic.

1.1.1 LAD contacts

Table 7-1 Normally open and normally closed contacts

LAD	Description
	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.

Supported Properties: Instruction Type

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.

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

Supported Properties: Instruction Type, Inputs Count

Table 7-4 Data types for the parameters

Parameter	Data type	Description
IN1, IN2	Bool	Input bit
OUT	Bool	Output bit

NOT logic inverter

Table 7-5 NOT Logic inverter

LAD	FBD	Description
		<p>For FBD programming, you can drag the "Bitwise inverting" instruction from the Catalog and then drop it on a network.</p> <p>The LAD NOT contact inverts the logical state of power flow input.</p> <ul style="list-style-type: none"> • 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.

Supported Properties: None

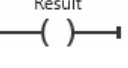
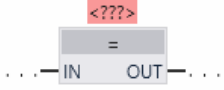
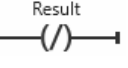
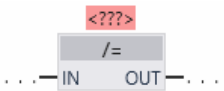
Table 7-6 Data types for the parameters

Parameter	Data type	Description
IN	Bool	Input bit
OUT	Bool	Output bit

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

LAD	FBD	Description
		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.
		

Supported Properties: Instruction Type

Table 7-8 Data types for the parameters

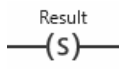
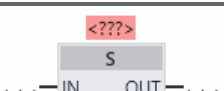
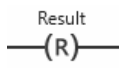

Parameter	Data type	Description
TOP (Argument above the instruction)	Bool	Assigned bit
OUT	Bool	Follows state of "IN"

- 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.

1.2 Set and reset instructions

Set and Reset 1 bit

Table 7-9 S and R instructions

LAD	FBD	Description
		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. These instructions can be placed anywhere in the network.
		

Supported Properties: Instruction Type

Table 7-10 Data types for the parameters

Parameter	Data type	Description
IN (or connect to contact/gate logic)	Bool	Bit location to be monitored
TOP (Argument above the instruction)	Bool	Bit location to be set or reset
OUT	Bool	Follows state of "IN"

Set-dominant and Reset-dominant bit latches

Table 7-11 RS and SR instructions

LAD/ FBD	Description
	RS 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.
	SR 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.

Supported Properties: None

Table 7-12 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
Q	Bool	Assigned bit output "Q"
INSTANCE (Argument above the instruction)	RS	RS and SR function block (system) instance

Instruction	S	R1	Q
RS	0	0	Previous state
	0	1	0
	1	0	1
	1	1	0
SR	S1	R	
	0	0	Previous state
	0	1	0
	1	0	1
	1	1	1

1.3 Positive and negative edge instructions

Table 7-13 P_TRIG and N_TRIG instructions Version 1.0

LAD	FBD	Description
		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).
		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).

Supported Properties: Instruction Type

Table 7-14 P_TRIG and N_TRIG instructions Version 2.0

LAD	FBD	Description
		<p>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.</p> <p>FBD: The output logic state is TRUE when a positive transition (OFF to ON) is detected on the assigned input bit. This is the version 2.0 of R_TRIG function block.</p>
		<p>LAD: The state of this contact is TRUE when a negative transition (ON to OFF) is detected on the assigned input bit. The contact logic state is then combined with the power flow in state to set the power flow out state. The N contact can be located anywhere in the network except the end of a branch.</p> <p>FBD: The output logic state is TRUE when a negative transition (ON to OFF) is detected on the assigned input bit. This is the version 2.0 of F_TRIG function block.</p>

Supported Properties: Instruction Type

Table 7-15 P_TRIG and N_TRIG instructions Version 3.0

LAD	FBD	Description
		<p>LAD: The assigned bit "Q" is TRUE when a positive transition (OFF to ON) is detected on the "CLK" input. The power flow in state always passes through the coil as the power flow out state.</p> <p>FBD: The assigned bit "Q" 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.</p>
		<p>LAD: The assigned bit "Q" is TRUE when a negative transition (ON to OFF) is detected on the "CLK" input. The power flow in state always passes through the coil as the power flow out state.</p> <p>FBD: The assigned bit "Q" 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.</p>

Supported Properties: Instruction Type

Table 7-16 Data types for the parameters

Parameter	Data type	Description
CLK	Bool	Power flow or input bit whose transition edge is to be detected
Q	Bool	Output which indicates an edge was detected
INSTANCE (Argument above the instruction)	R_TRIG, F_TRIG	R_TRIG and F_TRIG function block (system) instance

All edge instructions use a memory bit (M) in their instance 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.

<p>TIP</p> <p>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 (M) 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 instance for each edge instruction, and you should not use this bit any other place in your program. You</p>

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 G, or Static memory (in an OB or FB) for instantiating R_TRIG or F_TRIG.

💡 TIP

Sometimes you are planning for a multi task program that only on edge detection operation must be occur in individual task. That means when the first OB aware an edge detection, it must execute a routine and so other OBs do not. In this case you create a global R_TRIG or F_TRIG instance and assign it to multiple edge detection instructions in different OBs. So, you are sure that only one time an edge detection leads to execution a code block.

2. Word logic operations

2.1 AND, OR, and XOR instructions

Table 7-17 AND, OR, and XOR instruction

LAD/ FBD	Description
	AND: Logical AND
	OR: Logical OR
	XOR: Logical exclusive OR

Supported Properties: Instruction Type, Operation Data Type, Inputs Count

To increase or decrease inputs, click the "▲" or "▼" icon in the Properties pane.

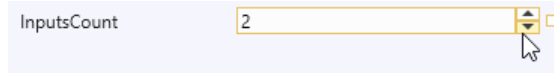


Table 7-18 Data types for the parameters

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

The data type selection sets parameters IN1, IN2, and OUT to the same data type. The corresponding bit values of IN1 and IN2 are combined to produce a binary logic result at parameter OUT. ENO is always TRUE following the execution of these instructions.

2.2 Invert instruction

Table 7-19 INV instruction

LAD/ FBD	Description
	Calculates the binary one's complement of the parameter IN. The one's complement is formed by inverting each bit value of the IN parameter (changing each 0 to 1 and each 1 to 0). ENO is always TRUE following the execution of this instruction.

Supported Properties: Operation Data Type

Table 7-20 Data types for the parameters

Parameter	Data type	Description
IN	Byte, Word, DWord, LWord	Data element to invert
OUT	Byte, Word, DWord, LWord	Inverted output

2.3 Shift and Rotate

2.3.1 Shift instructions

Table 7-21 SHR and SHL instructions

LAD/ FBD	Description
	<p>Use the shift instructions (SHL and SHR) to shift the bit pattern of parameter IN. The result is assigned to parameter OUT.</p> <p>Parameter N specifies the number of bit positions shifted:</p> <ul style="list-style-type: none"> SHR: Shift bit pattern right SHL: Shift bit pattern left

Supported Properties: Instruction Type, Operation Data Type

Table 7-22 Data types for the parameters

Parameter	Data type	Description
IN	Byte, Word, DWord, LWord	Bit pattern to shift
N	AnyInt	Number of bit positions to shift
OUT	Byte, Word, DWord, LWord	Bit pattern after shift operation

- For N=0, no shift occurs. The IN value is assigned to OUT.
- Zeros are shifted into the bit positions emptied by the shift operation.
- If the number of positions to shift (N) exceeds the number of bits in the target value (8 for Byte, 16 for Word, 32 for DWord), then all original bit values will be shifted out and replaced with zeros (zero is assigned to OUT).
- ENO is always TRUE for the shift operations.

Table 7-23 SHL example for Word data

Shift the bits of a Word to the left by inserting zeroes from the right (N = 1)			
IN	1110 0010 1010 1101	OUT value before first shift:	1110 0010 1010 1101
		After first shift left:	1100 0101 0101 1010
		After second shift left:	1000 1010 1011 0100
		After third shift left:	0001 0101 0110 1000

2.4 Rotate instructions

Table 7-24 ROR and ROL instructions

LAD/ FBD	Description
	<p>Use the rotate instructions (ROR and ROL) to rotate the bit pattern of parameter IN. The result is assigned to parameter OUT. Parameter N defines the number of bit positions rotated.</p> <ul style="list-style-type: none"> ROR: Rotate bit pattern right ROL: Rotate bit pattern left

Supported Properties: Instruction Type, Operation Data Type

Parameter	Data type	Description
IN	Byte, Word, DWord, LWord	Bit pattern to rotate
N	AnyInt	Number of bit positions to rotate
OUT	Byte, Word, DWord, LWord	Bit pattern after rotate operation

- For N=0, no rotate occurs. The IN value is assigned to OUT.
- Bit data rotated out one side of the target value is rotated into the other side of the target value, so no original bit values are lost.
- If the number of bit positions to rotate (N) exceeds the number of bits in the target value (8 for Byte, 16 for Word, 32 for DWord), then the rotation is still performed.
- ENO is always TRUE following execution of the rotate instructions.

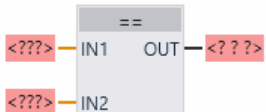
Table 7-25 ROR example for Word data

Rotate bits out the right -side into the left -side (N = 1)			
IN	0100 0000 0000 0001	OUT value before first rotate:	0100 0000 0000 0001
		After first rotate right:	1010 0000 0000 0000
		After second rotate right:	0101 0000 0000 0000

3. Comparison

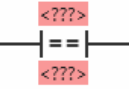
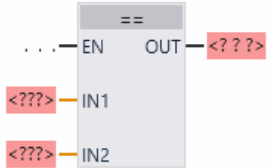
3.1 Compare

Table 7-26 Compare instructions Version 1.0

FBD	Description
	Compares two values of the same data type. When the FBD box comparison is TRUE, then the box output is TRUE.

Supported Properties: Instruction Type, Operation Data Type

Table 7-27 Compare instructions Version 2.0

LAD	FBD	Description
		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 (version 2.0) is TRUE, then the box output is TRUE.

Supported Properties: Instruction Type, Operation Data Type

For LAD and FBD: Click the InstructionType (such as "==") to change the comparison type and OperationDataType to change comparison data type from the drop-down list.

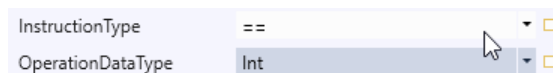


Table 7-28 Data types for the parameters

Parameter	Data type	Description
IN1, IN2	Any	Values to compare
Q	Bool	Comparison result

Table 7-29 Comparison descriptions

Relation type	Description
>	IN1 is greater than IN2
>=	IN1 is greater than or equal to IN2
==	IN1 is equal to IN2

<	IN1 is less than IN2
<=	IN1 is less than or equal to IN2
<>	IN1 is not equal to IN2

3.2 In-range and Out-of-range instructions

Table 7-30 In Range and Out of Range instructions

LAD/ FBD	Description
	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.

Supported Properties: Instruction Type, Operation Data Type

Table 7-31 Data types for the parameters

Parameter	Data type	Description
MIN, VAL, MAX	AnyNum	Comparator inputs

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

4. Math

4.1 Add, subtract, multiply and divide instructions

Table 7-32 Add, subtract, multiply and divide instructions

LAD/ FBD	Description
	<ul style="list-style-type: none"> • ADD: Addition (IN1 + IN2 = OUT) • SUB: Subtraction (IN1 - IN2 = OUT) • MUL: Multiplication (IN1 * IN2 = OUT) • DIV: Division (IN1 / IN2 = OUT) <p>An Integer division operation truncates the fractional part of the quotient to produce an integer output.</p>

Supported Properties: Instruction Type, Operation Data Type, Inputs Count

Table 7-33 Data types for the parameters

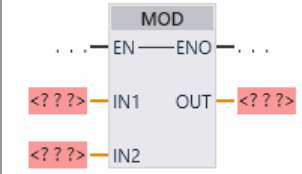
Parameter	Data type	Description
IN1, IN2	AnyNum	Math operation inputs
OUT	AnyNum	Math operation output

Table 7-34 ENO status

ENO	Description
True	No error
False	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.
	Division by 0 (IN2 = 0): The result is undefined and zero is returned.
	Real/LReal: If one of the input values is NaN (not a number) then NaN is returned.
	MUL Real/LReal: If one IN value is zero and the other is INF, this is an illegal operation and NaN is returned.
	DIV Real/LReal: If both IN values are zero or INF, this is an illegal operation and NaN is returned.

4.2 Modulo instruction

Table 7-35 MOD instruction

LAD/ FBD	Description
	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.

Supported Properties: Operation Data Type

Table 7-36 Data types for parameters

Parameter	Data type	Description
IN1, IN2	AnyInt	Modulo inputs
OUT	AnyInt	Modulo output

General exponentiation instruction

Table 7-37 EXPT instruction

LAD/ FBD	Description
	General exponentiation is an operation involving two numbers, the base and the exponent or power. You can use the EXPT instruction to raising IN1 to a power of IN2 and return the result at the OUT output.

Supported Properties: Input1 Data Type

Table 7-38 Data types for parameters

Parameter	Data type	Description
IN1	AnyReal	Number to be raised
IN2	AnyNum	Number to be powered to
OUT	AnyInt	Exponent output

4.3 Absolute value instruction

Table 7-39 ABS instruction

LAD/ FBD	Description
	Calculates the absolute value of a signed integer or real number at parameter IN and stores the result in parameter OUT.

Supported Properties: Operation Data Type

Table 7-40 Data types for parameters

Parameter	Data type	Description
IN	AnyNum	Math operation input
OUT	AnyNum	Math operation output

4.4 Increment and decrement instructions

Table 7-41 INC and DEC instructions

LAD/ FBD	Description
	Increments a signed or unsigned integer number value: IN/OUT value + 1 = IN/OUT value
	Decrements a signed or unsigned integer number value: IN/OUT value - 1 = IN/OUT value

Supported Properties: Instruction Type, Operation Data Type

Table 7-42 Data types for parameters

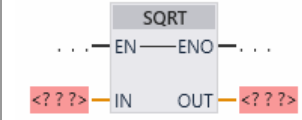
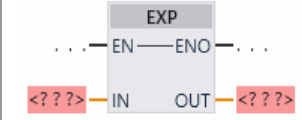
Parameter	Data type	Description
IN/ OUT	AnyInt	Math operation input and output

4.5 Floating-point math instructions

You use the floating-point instructions to program mathematical operations using a Real or LReal data type:

- SQRT: Square root ($\sqrt{\text{IN}} = \text{OUT}$)
- LN: Natural logarithm ($\text{LN}(\text{IN}) = \text{OUT}$)
- LOG: Logarithm to base 10 ($\text{LOG}(\text{IN}) = \text{OUT}$)
- EXP: Natural exponential ($e^{\text{IN}} = \text{OUT}$), where base $e = 2.71828182845904523536$
- SIN: Sine ($\sin(\text{IN radians}) = \text{OUT}$)
- COS: Cosine ($\cos(\text{IN radians}) = \text{OUT}$)
- TAN: Tangent ($\tan(\text{IN radians}) = \text{OUT}$)
- ASIN: Inverse sine ($\arcsin(\text{IN}) = \text{OUT radians}$), where the $\sin(\text{OUT radians}) = \text{IN}$
- ACOS: Inverse cosine ($\arccos(\text{IN}) = \text{OUT radians}$), where the $\cos(\text{OUT radians}) = \text{IN}$
- ATAN: Inverse tangent ($\arctan(\text{IN}) = \text{OUT radians}$), where the $\tan(\text{OUT radians}) = \text{IN}$

Table 7-43 Examples of floating-point math instructions

LAD/ FBD	Description
	Square root: $\sqrt{\text{IN}} = \text{OUT}$ For example: If $\text{IN} = 81$, then $\text{OUT} = 9$.
	Natural exponential: $e^{\text{IN}} = \text{OUT}$ For example: If $\text{IN} = 3$, then $\text{OUT} = 20.0855$.

Supported Properties: Instruction Type, Operation Data Type

Table 7-44 Data types for parameters

Parameter	Data type	Description
IN	AnyReal	Math operation input
OUT	AnyReal	Math operation output

5. Timer and Counter

5.1 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 structure to store timer data that is specified at the top of the box or coil instruction.

Table 7-45 Timer instructions

LAD/ FBD	Description
	The TON timer sets output Q to ON after a preset time delay.
	The TOF timer resets output Q to OFF after a preset time delay.
	The TP timer generates a pulse with a preset width time.

Supported Properties: Instruction Type

Table 7-46 Data types for the parameters

Parameter	Data type	Description
IN	Bool	TP and TON: FBD: 0=Disable timer, 1=Enable timer LAD: No power flow=Disable timer, Power flow=Enable timer TOF: FBD: 0=Enable timer, 1=Disable timer LAD: No power flow=Enable timer, Power flow=Disable timer
PT	Time	Preset time input
Q	Bool	Q box output
ET	Time	Elapsed time

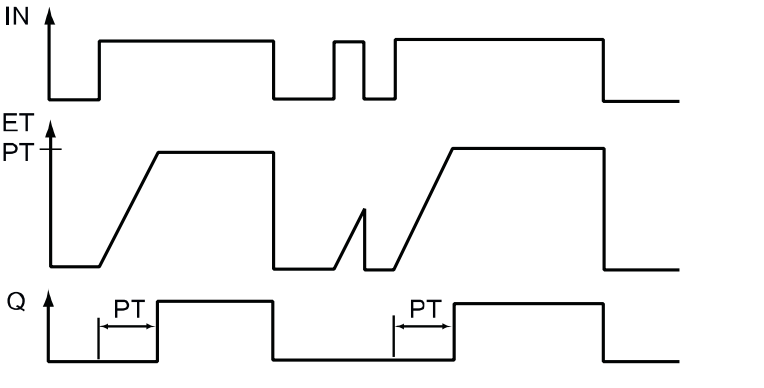
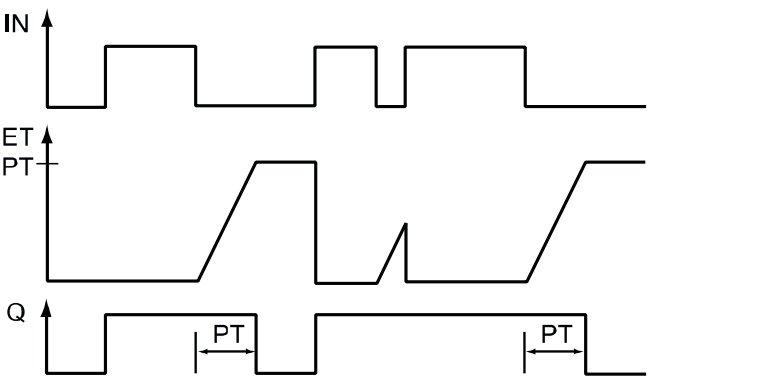
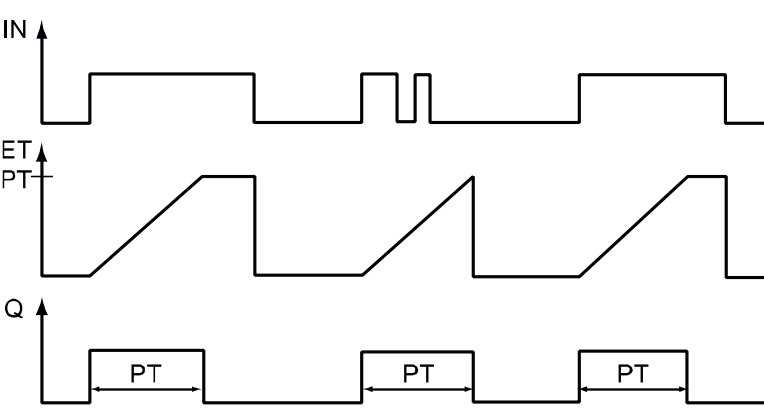
Table 7-47 Effect of value changes in the PT and IN parameters

Timer	Changes in the PT and IN parameters
TON	<ul style="list-style-type: none"> Changing PT is considered while the timer runs. Changing IN to FALSE, while the timer runs, resets and stops the timer.
TOF	<ul style="list-style-type: none"> Changing PT is considered while the timer runs. Changing IN to TRUE, while the timer runs, resets and stops the timer.
TP	<ul style="list-style-type: none"> Changing PT is considered while the timer runs. Changing IN has no effect while the timer runs.

PT (preset time) and ET (elapsed time) values are stored in the specified IEC_TIMER structures 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) and as compound time units like T#2s200ms. The negative range of the TIME data type 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.

5.1.1 Operation of the timers

Table 7-48 Types of IEC timers

Timer	Changes in the PT and IN parameters
<p>TON: ON-delay timer The TON timer sets output Q to ON after a preset time delay.</p>	
<p>TOF: OFF-delay timer The TOF timer resets output Q to OFF after a preset time delay.</p>	
<p>TP: Pulse timer The TP timer generates a pulse with a preset width time.</p>	

In the CPU, no dedicated resource is allocated to any specific timer instruction. Instead, each timer utilizes its own timer structure in 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 TON, TOF or TP instruction, the value of the continuously-running internal CPU timer is copied into the STIME member of the 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 ET and Q members are then updated in the 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 (TON, TOF or TP) is executed
- The "ET" member of the timer structure is referenced directly by an instruction
- The "Q" member of the timer structure is referenced directly by an instruction

5.1.2 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 (TON, TOF, TP) is executed. 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.
- 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 ET 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 ET or Q from the timer structure as a parameter.
- Although not typical, you can assign the same timer structure to multiple timer instructions. In general, to avoid unexpected interaction, you should only use one timer instruction (TON, TOF, TP) per 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, use the "Q" member of the timer structure as the parameter for the normally-closed contact in front of the timer instruction.

5.1.3 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 (TON, TOF and TP 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 which cannot be made retentive. To make your timer data retentive, you must use a global retained instance of that timer.

5.1.4 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 instance of a timer in a reference tag table editor (G).
- 2- In the "Retain" column, check the box so that the timer structure will be retentive. Repeat this process to create structures for all the timers that you want to store as retentive timers. 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- On the top of the new timer instruction, type the name (you can use the helper to browse) of the global instance structure that you created above (example: " StaticTON1").

5.2 Counters

Table 7-49 Counter instructions

LAD/ FBD	Description
	<p>Use the counter instructions to count internal program events and external process events. Each counter uses a structure stored in a system structure to maintain counter data. You assign the data block when the counter instruction is placed in the editor.</p> <ul style="list-style-type: none"> • CTU is a count-up counter • CTD is a count-down counter • CTUD is a count-up-and-down counter

Supported Properties: Instruction Type

TIP

The default counter structures operate as Int data type. For other type of integers select another instruction from Catalog pane. Example: to have a UInt counter you should use CTU_UDINT instruction.

Table 7-50 Data types for the parameters

Parameter	Data type	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	AnyInt	Preset count value
Q, QU	Bool	True if CV \geq PV
QD	Bool	True if CV \leq 0
CV	AnyInt	Current count value

NOTICE

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.

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.

TIP

For faster counting operations you should use hardware counters. For more info about the hardware counters of your device see its technical data.

5.2.1 Operation of the counters

Table 7-51 Operation of the CTU counter

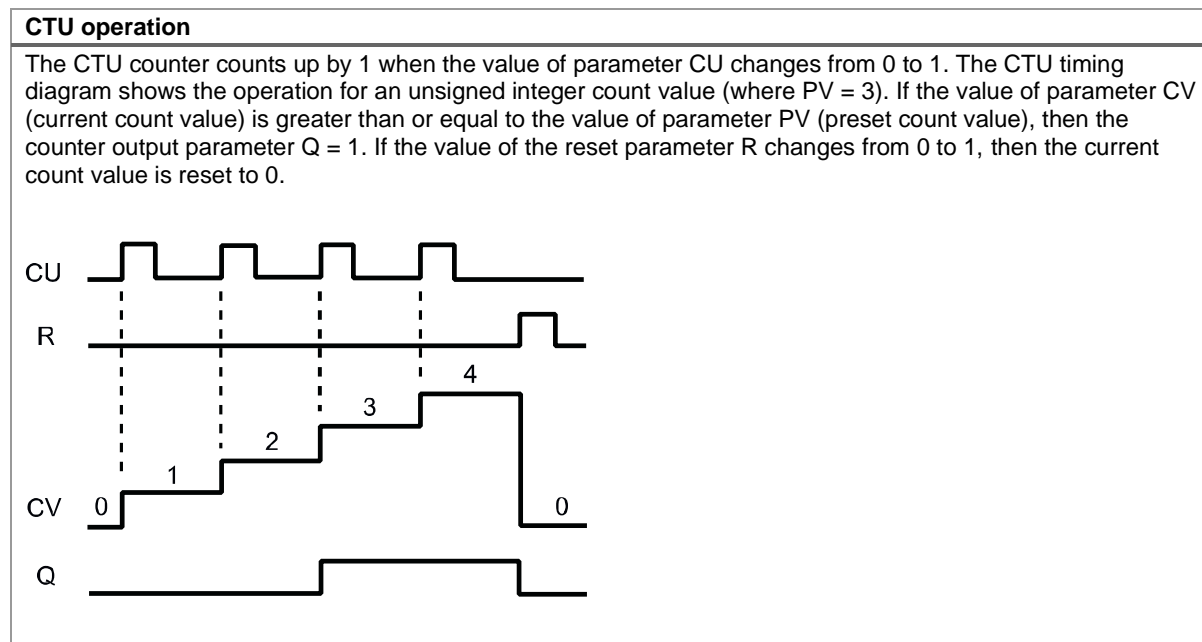


Table 7-52 Operation of the CTD counter

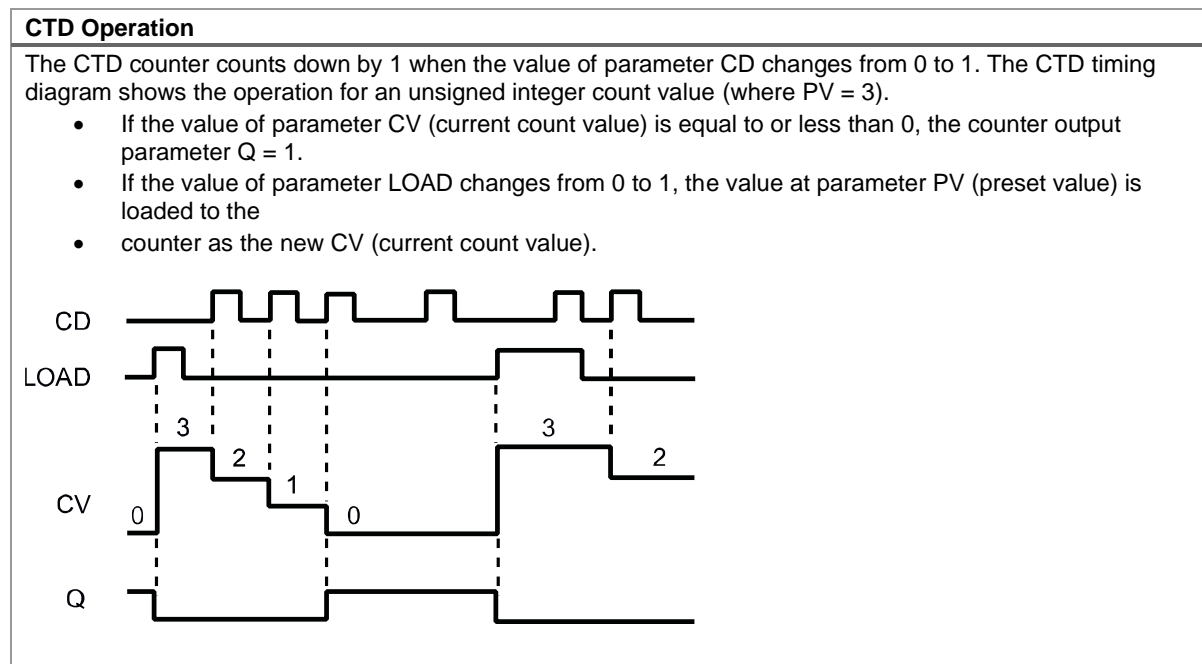
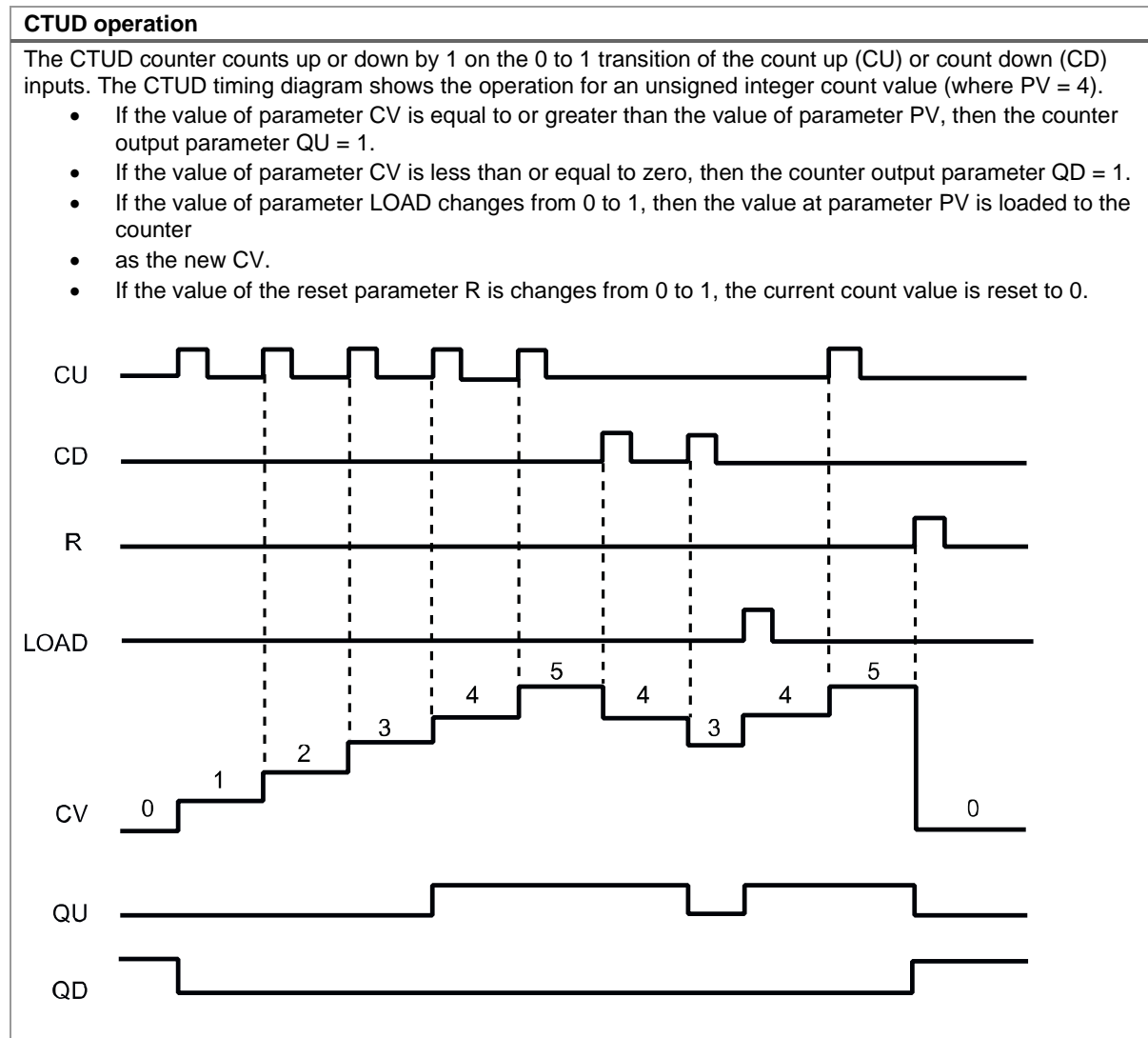


Table 7-53 Operation of the CTUD counter



5.2.2 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 use a global retained instance of that counter.

5.2.3 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 instance of a counter in a reference tag table editor (G). 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
-------------------	--

CTU	INT
CTU_DINT	DINT
CTU_LINT	LINT
CTU_UDINT	UDINT

CTU_ULINT	ULINT
CTD	INT
CTD_DINT	DINT
CTD_LINT	LINT
CTD_UDINT	UDINT
CTD_ULINT	ULINT
CTUD	INT
CTUD_DINT	DINT
CTUD_LINT	LINT
CTUD_UDINT	UDINT
CTUD_ULINT	ULINT

- 2- In the "Retain" column, check the box so that the counter structure will be retentive. Repeat this process to create structures for all the counters that you want to store as retentive counters. Rename the counter structures if desired.
- 3- Open the program block for editing where you want to place a retentive counter (OB, FC, or FB).
- 4- Place the counter instruction at the desired location.
- 5- On the top of the new counter instruction, type the name (you can use the helper to browse) of the global instance structure that you created above (example: " StaticCTU1").

6. Moving and conversion

6.1 Move instructions

Use the Move instructions to copy data elements to a new memory. The source data is not changed by the move process.

- The MOVE instruction copies a single data element (Any) from the source address specified by the IN parameter to the destination addresses specified by the OUT parameter.
- The VAR_MOVE instruction copies a data element by its pointer (Variant) from the source address specified by the IN parameter to the destination addresses specified by the OUT parameter.

Table 7-54 MOVE instructions

LAD/ FBD	Description
	Copies a data element stored at a specified address to a new address or multiple addresses.
	Copies a data element stored at a specified address to a new address by its pointer or multiple addresses.
	copies a block of data elements to a new block.

Supported Properties: Outputs Count (Except BLK_MOVE)

Table 7-55 Data types for the MOVE instruction

Parameter	Data type	Description
IN	Any	Source address
OUT	Any	Destination address

Table 7-56 Data types for the VAR_MOVE instruction

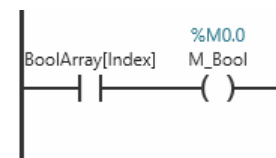
Parameter	Data type	Description
IN	Variant	Source address
OUT	Variant	Destination address

Table 7-57 Data types for the BLK_MOVE instruction

Parameter	Data type	Description
IN	Variant	Source element
BUF	Byte[*]	Moving buffer
OUT	Variant	Destination element
ERROR	Bool	True if moving encounter an error

6.2 Accessing data by array indexing

To access elements of an array with a variable, simply use the variable as an array index in your program logic. For example, the following network sets an output based on the Boolean value of an array of Booleans in "BoolArray" referenced by the PLC tag "Index".



6.3 Convert instruction

Table 7-58 Convert instruction

LAD/ FBD	Description
	Converts a data element from one data type to another data type.

Supported Properties: Operation Data Type

Table 7-59 Data types for the parameters

Parameter	Data type	Description
IN	Any	Input value
OUT	Any	Input value converted to a new data type

NOTICE

When you convert a tag value to an AnyBit data type tag (such as byte, word, DWord, LWord), the data will be memory copied to that AnyBit tag and vice versa. Example: if you convert 3.1415 (Real) to a DWord tag (Destination) the result will be: 1078529622 (16#40490e56)

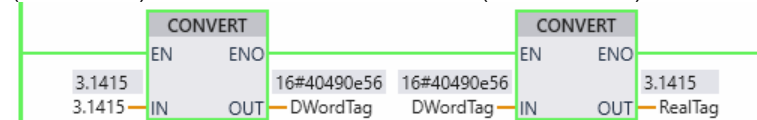


Table 7-60 ENO status

ENO	Description
True	No error
False	Conversion error

6.4 BCD conversion instructions

Table 7-61 BCD conversion instructions

LAD/ FBD	Description
	Converts a BCD format data element to an AnyUnsigned data type.
	Converts an AnyUnsigned data type to a BCD format data element.

Supported Properties: Operation Data Type

Table 7-62 Data types for the BCD_TO instruction

Parameter	Data type	Description
IN	AnyBit	Input value
OUT	AnyUnsigned	Input value converted to a new data type

Table 7-63 Data types for the TO_BCD instruction

Parameter	Data type	Description
IN	AnyUnsigned	Input value
OUT	AnyBit	Input value converted to a new data type


Table 7-64 ENO status

ENO	Description
True	No error
False	Conversion error

6.5 Round, ceiling, floor and truncate instructions

Table 7-65 ROUND, CEIL, FLOOR and TRUNC instructions

LAD/ FBD	Description
	<p>Converts a real number to an integer. The default data type is Int. The real number fraction is rounded to the nearest integer value (IEEE - round to nearest). If the number is exactly one-half the span between two integers (for example, 10.5), then the number is rounded to the Nearest whole number. For example:</p> <ul style="list-style-type: none"> ROUND (10.5) = 11 ROUND (11.5) = 12
	Converts an AnyReal number (Real or LReal) to the closest integer greater than or equal to the selected real number (IEEE "round to +infinity").
	Converts an AnyReal number (Real or LReal) to the closest integer smaller than or equal to the selected real number (IEEE "round to -infinity").

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

Supported Properties: Instruction Type, Operation Data Type

Table 7-66 Data types for the parameters


Parameter	Data type	Description
IN	AnyReal	Floating point input
OUT	AnyInt	Converted output

Table 7-67 ENO status

ENO	Description
True	No error
False	Conversion error

6.6 Swap instruction

Table 7-68 SWAP instruction

LAD/ FBD	Description
	<p>Reverses the byte order for two-byte, four-byte and eight-byte data elements. No change is made to the bit order within each byte.</p>

Supported Properties: Operation Data Type

Table 7-69 Data types for the parameters

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

Example 1	Parameter IN = %MB0 (before execution)		Parameter OUT = %MB4, (after execution)	
Address	%MB0	%MB1	%MB4	%MB5
16#1234	12	34	34	12
Word	MSB	LSB	MSB	LSB

Example 2	Parameter IN = %MB0 (before execution)				Parameter OUT = %MB4, (after execution)			
Address	%MB0	%MB1	%MB2	%MB3	%MB4	%MB5	%MB6	%MB7
16#12345678	12	34	56	78	78	56	34	12
DWord	MSB			LSB	MSB			LSB

6.7 Serialize instruction

Table 7-70 SERIALIZE instruction

LAD/ FBD	Description
	<p>You can use the "Serialize" instruction to convert several PLC data types (UDT), STRUCT or ARRAY of <data type> to a sequential representation without losing parts of their structure.</p> <p>You use the instruction to temporarily save multiple structured data items from your program in a buffer, which should preferably be in an array of byte, and send them to another CPU or network.</p>

Supported Properties: None

Table 7-71 Data types for the parameters

Parameter	Data type	Description
INDEX	AnyInt	Start position of byte array destination buffer
IN	Variant	Data to be serialized
OUT	Byte[*]	Serialized stream byte array destination

Table 7-72 ENO status

ENO	Description
True	No error
False	Conversion error

NOTICE

The maximum serializable data length is 1024.

6.8 Deserialize instruction

Table 7-73 DESERIALIZE instruction

LAD/ FBD	Description
	<p>You can use the "Deserialize" instruction to convert back the sequential representation of a User data type (UDT), STRUCT or ARRAY of <data type> and to fill its entire contents.</p>

Supported Properties: None

Table 7-74 Data types for the parameters

Parameter	Data type	Description
INDEX	AnyInt	Start position of byte array source buffer
SOURCE	Byte[*]	Source byte array
OUT	Variant	Deserialized data

Table 7-75 ENO status

ENO	Description
True	No error
False	Conversion error

7. Program Control

7.1 FOR statement

Table 7-76 FOR statement instruction

LAD/ FBD	Description
	A FOR statement is used to repeat a sequence of networks as long as a control variable is within the specified range of values. The definition of a loop with FOR includes the specification of an initial and an end value. Both values must be the same type as the control variable.

Supported Properties: None

Table 7-77 Parameters

Parameter	Data type	Description
FROM	AnyInt	Required. Simple expression (tag or constant) that specifies the initial value of the control variables
TO	AnyInt	Required. Simple expression (tag or constant) that determines the final value of the control variables
BY	AnyInt	Required. Amount by which an "OUT" is changed after each loop. The "BY" has the same data type as "OUT"
OUT	AnyInt	Optional. A tag that serves as a loop counter

The FOR statement executes as follows:

- At the start of the loop, the control variable (OUT) is set to the initial value (FROM) and each time the loop iterates, it is incremented by the specified increment (positive increment) or decremented (negative increment) until the final value is reached.
- Following each run through of the loop, the condition is checked (final value reached) to establish whether or not it is satisfied. If the condition is satisfied, the sequence of statements is executed, otherwise the loop and with it the sequence of statements is skipped.
- The execution scope of a for statement is the networks between the FOR statement and the END statement. If you put the FOR and END statements in the same network, that network will be the execution scope of FOR.
- You can use nested FOR loops by placing a FOR statement inside another FOR and before its END statement.

Example:

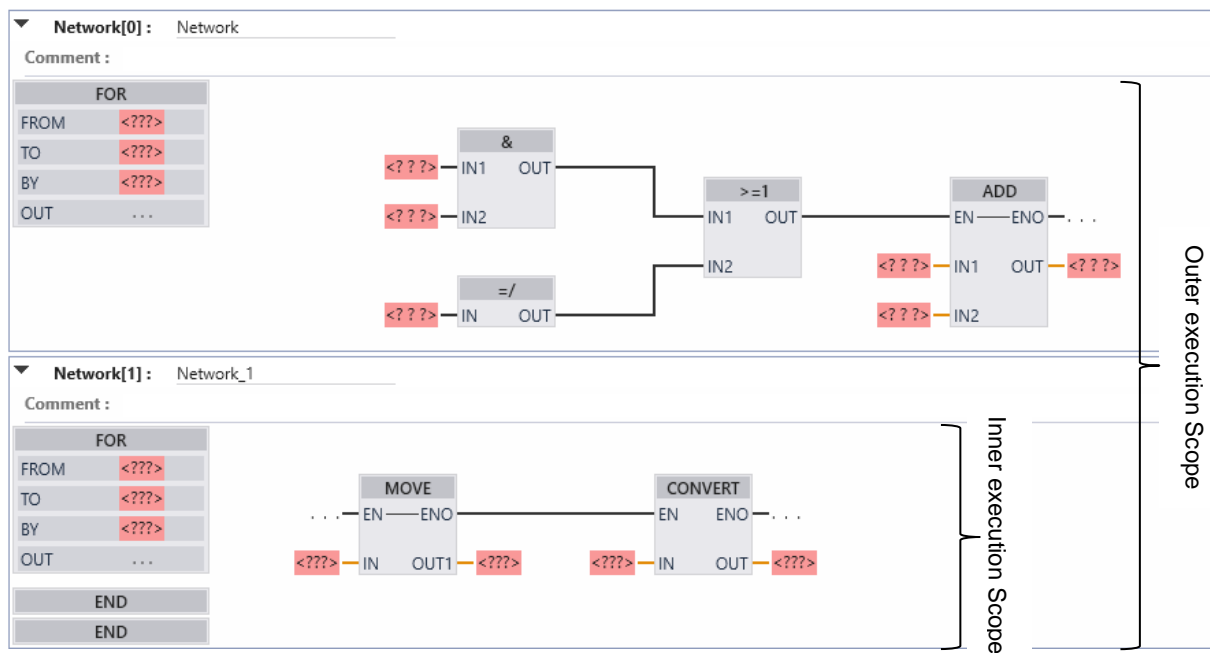


Figure 7-3 An example of nested FOR statements

7.2 WHILE statement

Table 7-78 WHILE statement instruction

LAD/ FBD	Description
	<p>The WHILE statement performs a series of statements until a given condition (IN) is True.</p> <p>You can nest WHILE loops. The END statement refers to the last executed WHILE instruction.</p>

Supported Properties: None

Table 7-79 Parameters

Parameter	Data type	Description
IN	Bool	Required. A Bool tag that evaluates to True or False.

The WHILE statement executes according to the following rules:

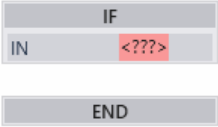
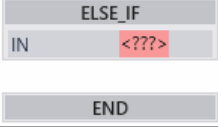

- Prior to each iteration of the loop body, the execution condition is evaluated.
- Once the value FALSE occurs, the loop is skipped and the statement following the loop is executed

WARNING Always be careful when using the WHILE statement. You must plan a condition for IN parameter to be finally set to False. If the condition (IN) will be set to always True, then the CPU will get stuck in an infinity loop.

7.3 IF statement

The IF statement is a conditional statement that controls program flow by executing a group of statements, based on the evaluation of a Bool value of a logical expression. You can also nest or structure the execution of multiple IF-ELSE statements.

Table 7-80 Elements of the IF statement

LAD/ FBD	Description
	<p>If condition (IN) is True, then execute the following statements until encountering the END statement.</p> <p>If condition (IN) is False, then skip to END statement (unless the program includes optional ELSE_IF or ELSE statements).</p>
	<p>The optional ELSE_IF statement provides additional conditions to be evaluated. For example: If condition (IN) in the IF statement is False, then the program evaluates condition-n (IN). If condition-n is True, then execute statement_N.</p>
	<p>The optional ELSE statement provides statements to be executed when the condition (IN) of the IF statement is False.</p>

Supported Properties: None

Table 7-81 Parameters

Parameter	Data type	Description
IN	Bool	Required. A Bool tag that evaluates to True or False.

💡 TIP

You can include multiple ELSE_IF statements within one IF statement.

An IF statement is executed according to the following rules:

- The first sequence of statements whose logical expression = True is executed. The remaining sequences of statements are not executed.
- If no Boolean expression = True, the sequence of statements introduced by ELSE is executed (or no sequence of statements if the ELSE branch does not exist).
- Any number of ELSE_IF statements can exist.

💡 TIP


Using one or more ELSIF branches has the advantage that the logical expressions following a valid expression are no longer evaluated in contrast to a sequence of IF statements. The runtime of a program can therefore be reduced.

7.4 RET execution control instruction

The optional RET instruction is used to terminate the execution of the current block. If and only if the RET input IN is true, then program execution of the current block will end at that point and instructions beyond the RET instruction will not be executed. If the current block is an OB its execution routine will be terminated until the next scan. If the current block is a FC or FB, its execution routine will be terminated until the next calling of that program block.

You are not required to use a RET instruction as the last instruction in a block; this is done automatically for you. You can have multiple RET instructions within a single block.

Table 7-82 RET execution control instruction

LAD/ FBD	Description
	<p>Terminates the execution of the current block</p>

Supported Properties: None

Table 7-83 Parameters

Parameter	Data type	Description
IN	Bool	Trigger for termination a program block

⚠ WARNING

You should keep in mind that all program control instructions prevent part of the program from running unless certain conditions are met. If there is a structural error in that part of the program that is supposed to be executed under certain conditions, the CPU may encounter a runtime error that has never been seen before executing that code.
 Example: the following code will not make an index out of range exception until the Condition tag value remains False.

```

    IF
    IN #Condition
    END

    MOVE
    EN ENO
    -1 IN OUT1 Index
    True IN OUT1 Array[Index]
    
```

8. Selection

8.1 Select

Table 7-84 SEL instruction

LAD/ FBD	Description
	<p>Depending on a switch (G input), the "Select" instruction selects one of the inputs, IN1 or IN2 and copies its content to the OUT output. When the input G has signal state False, the value at the input IN1 is moved. When the input G has signal state True, the value at the input IN2 is moved to the output OUT.</p> <p>All tags at all parameters must have the same data type.</p>

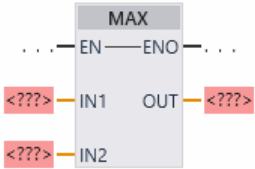
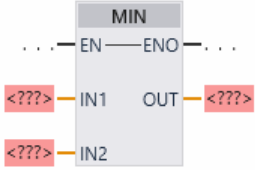
Supported Properties: None

Table 7-85 Data types for the parameters

Parameter	Data type	Description
G	Bool	Switch
IN1	Any	First input value
IN2	Any	Second input value
OUT	Any	Result

8.2 Get maximum and minimum

Table 7-86 MAX and MIN instructions

LAD/ FBD	Description
	The MAX instruction compares the value of two parameters IN1 and IN2 and assigns the maximum (greater) value to parameter OUT.
	The MIN instruction compares the value of two parameters IN1 and IN2 and assigns the minimum (lesser) value to parameter OUT.

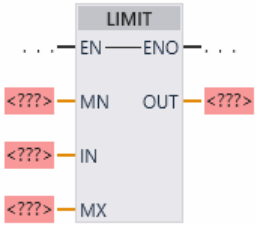
Supported Properties: Instruction Type

Table 7-87 Data types for the parameters

Parameter	Data type	Description
IN1	Any	First input value
IN2	Any	Second input value
OUT	Any	Result

8.3 Limit instruction

Table 7-88 LIMIT instruction

LAD/ FBD	Description
	<p>The Limit instruction tests if the value of parameter IN is inside the value range specified by parameters MN and MX and if not, clamps the value at MN or MX.</p> <p>If the value of parameter IN is within the specified range, then the value of IN is stored in parameter OUT. If the value of parameter IN is outside of the specified range, then the OUT value is the value of parameter MN (if the IN value is less than the MN value) or the value of parameter MX (if the IN value is greater than the MX value)</p>

Supported Properties: None

Table 7-89 Data types for the parameters

Parameter	Data type	Description
MN	Any	Minimum value
IN	Any	Input value
MX	Any	Maximum value
OUT	Any	Result

8.4 Multiplex instruction

Table 7-90 MUX instruction

LAD/ FBD	Description
	MUX copies one of many input values to parameter OUT, depending on the parameter K value.

Supported Properties: Inputs Count

Table 7-91 Data types for the parameters

Parameter	Data type	Description
K	AnyInt	<ul style="list-style-type: none"> 0 selects IN1 1 selects IN2 n selects INn
IN1,IN2,...INn	Any	Inputs
OUT	Any	Output

Table 7-92 ENO status

ENO	Description
True	No error
False	Index (K) out of range

8.5 Check for nullity

Table 7-93 IS_NULL instruction

LAD/ FBD	Description
	You can use this instruction to query whether the Variant or the reference points to a NULL pointer and therefore does not point to an object.

Supported Properties: None

Table 7-94 Data types for the parameters

Parameter	Data type	Description
IN	Variant	Input pointer
Q	Bool	Result

8.6 Check for array

Table 7-95 IS_ARRAY instruction

LAD/ FBD	Description
	You can use this instruction to query whether the Variant points to a tag of the Array data type.

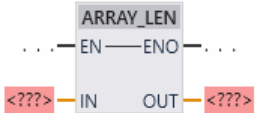
Supported Properties: None

Table 7-96 Data types for the parameters

Parameter	Data type	Description
IN	Variant	Input pointer
Q	Bool	Result

8.7 Get array length

Table 7-97 ARRAY_LEN instruction

LAD/ FBD	Description
	<p>You can use this instruction to query number of elements in an array.</p> <p>Example: an array defined as Bool[1,2,3] has a length of $1*2*3=6$</p>

Supported Properties: None

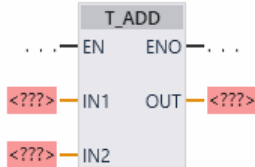
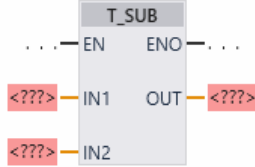
Table 7-98 Data types for the parameters

Parameter	Data type	Description
IN	Variant	Array pointer
OUT	AnyInt	Array length

9. Time

9.1 Time add and subtract

Table 7-99 T_ADD and T_SUB instructions

LAD/ FBD	Description
	<p>T_ADD adds the input IN1 Time value with the input IN2 Time value. Parameter OUT provides the Time value result.</p>
	<p>T_SUB subtracts the IN2 Time value from IN1 Time value. Parameter OUT provides the difference value as a Time data type.</p>

Supported Properties: Instruction Type

Table 7-100 Data types for the parameters

Parameter	Data type	Description
IN1	Time	Time value
IN2	Time	Time value to add or subtract
OUT	Time	Time sum or difference

Table 7-101 ENO status

ENO	Description
True	No error
False	Result out of range

9.2 Time multiplication and division

Table 7-102 T_MUL and T_DIV instructions

LAD/ FBD	Description
	T_MUL multiplies the input IN1 Time value in IN2 value. Parameter OUT provides the Time value result.
	T_DIV divides the input IN1 Time by IN2 value. Parameter OUT provides the Time value result.

Supported Properties: Instruction Type, Operation Data Type

Table 7-103 Data types for the parameters

Parameter	Data type	Description
IN1	Time	Time value
IN2	AnyNum	Multiply or device factor
OUT	Time	Time sum or difference

Table 7-104 ENO status

ENO	Description
True	No error
False	Result out of range

9.3 Time of day addition and subtraction time

Table 7-105 TOD_T_ADD and TOD_T_SUB instructions

LAD/ FBD	Description
	TOD_T_ADD adds the input IN1 TimeOfDay value with the input IN2 Time value. Parameter OUT provides the TimeOfDay value result.
	TOD_T_SUB subtracts the IN2 Time value from IN1 TimeOfDay value. Parameter OUT provides the difference value as a TimeOfDay data type.

Supported Properties: Instruction Type

Table 7-106 Data types for the parameters

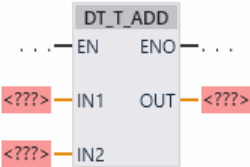
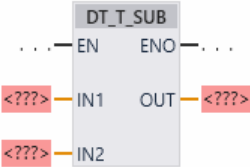
Parameter	Data type	Description
IN1	TimeOfDay	Time of day value
IN2	Time	Time value to add or subtract
OUT	TimeOfDay	Time of day sum or difference

Table 7-107 ENO status

ENO	Description
True	No error
False	Result out of range

9.4 Date addition and subtraction time

Table 7-108 DT_T_ADD and DT_T_SUB instructions

LAD/ FBD	Description
	DT_T_ADD adds the input IN1 DateTime value with the input IN2 Time value. Parameter OUT provides the DateTime value result.
	DT_T_SUB subtracts the IN2 Time value from IN1 DateTime value. Parameter OUT provides the difference value as a DateTime data type.

Supported Properties: Instruction Type

Table 7-109 Data types for the parameters

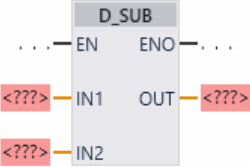
Parameter	Data type	Description
IN1	DateTime	Time of day value
IN2	Time	Time value to add or subtract
OUT	DateTime	Time of day sum or difference

Table 7-110 ENO status

ENO	Description
True	No error
False	Result out of range

9.5 Date subtraction

Table 7-111 D_SUB instruction

LAD/ FBD	Description
	D_SUB subtracts the IN2 Date value from IN1 Date value. Parameter OUT provides the difference value as a Time data type.

Supported Properties: None

Table 7-112 Data types for the parameters

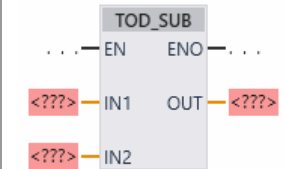
Parameter	Data type	Description
IN1	Date	Date value
IN2	Date	Date value to subtract
OUT	Time	Time difference

Table 7-113 ENO status

ENO	Description
True	No error
False	Result out of range

9.6 Time of day subtraction

Table 7-114 TOD_SUB instruction

LAD/ FBD	Description
	TOD_SUB subtracts the IN2 TimeOfDay value from IN1 TimeOfDay value. Parameter OUT provides the difference value as a Time data type.

Supported Properties: None

Table 7-115 Data types for the parameters

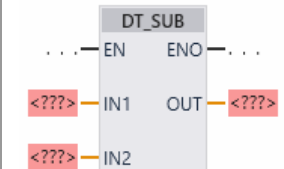
Parameter	Data type	Description
IN1	TimeOfDay	Time of day value
IN2	TimeOfDay	Time of day value to subtract
OUT	Time	Time difference

Table 7-116 ENO status

ENO	Description
True	No error
False	Result out of range

9.7 Date and time subtraction

Table 7-117 DT_SUB instruction

LAD/ FBD	Description
	DT_SUB subtracts the IN2 DateTime value from IN1 DateTime value. Parameter OUT provides the difference value as a Time data type.

Supported Properties: None

Table 7-118 Data types for the parameters

Parameter	Data type	Description
IN1	DateTime	Date and time value


IN2	DateTime	Date and time value to subtract
OUT	Time	Time difference

Table 7-119 ENO status

ENO	Description
True	No error
False	Result out of range

9.8 Time concatenation

Table 7-120 CONCAT_D_TOD instruction

LAD/ FBD	Description
	CONCAT_D_TOD concatenates the IN2 TimeOfDay value to IN1 Date value. Parameter OUT provides the concatenated value as a DateTime data type.

Supported Properties: None

Table 7-121 Data types for the parameters

Parameter	Data type	Description
IN1	Date	Date value
IN2	TimeOfDay	Time of day value to concatenate
OUT	DateTime	DateTime result

Table 7-122 ENO status

ENO	Description
True	No error
False	Result out of range

10. Character and string

10.1 String data overview

String data is stored as a 64 bytes of ASCII character codes. The number of stored bytes occupied by the String format is always 64 bytes.

String input and output data must be initialized as valid strings in memory, before execution of any string instructions.

10.2 String operation instructions

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

10.2.1 LEN

Table 7-123 Length instruction

LAD/ FBD	Description
	<p>LEN (length) provides the current length of the string IN at output OUT. An empty string has a length of zero.</p>

Supported Properties: None

Table 7-124 Data types for the parameters

Parameter	Data type	Description
IN	String	Input string
OUT	Int	String length

10.2.2 LEFT and RIGHT

Table 7-125 Left and right substring operations

LAD/ FBD	Description
	<p>LEFT (Left substring) provides a substring made of the first L characters of string parameter IN.</p>
	<p>RIGHT (Right substring) provides the last L characters of a string.</p>

Supported Properties: Instruction Type

Table 7-126 Data types for the parameters

Parameter	Data type	Description
IN	String	Input string
L	AnyInt	Length of the substring to be created: <ul style="list-style-type: none"> • LEFT uses the left-most characters number of characters in the string • RIGHT uses the right-most number of characters in the string
OUT	String	Output string

Table 7-127 ENO status

ENO	Description
True	No error
False	If L is greater than the current length of the IN string

10.2.3 MID

Table 7-128 middle substring operation

LAD/ FBD	Description
	MID (Middle substring) provides the middle part of a string. The middle substring is L characters long and starts at character position P (inclusive).

Supported Properties: None

Table 7-129 Data types for the parameters

Parameter	Data type	Description
IN	String	Input string
L	AnyInt	Length of the substring to be created. It uses the number of characters starting at position P within the string
P	AnyInt	Position of first substring character to be copied P= 1, for the initial character position of the IN string
OUT	String	Output string

Table 7-130 ENO status

ENO	Description
True	No error
False	If the sum of L and P exceeds the current length of the string parameter IN

10.2.4 CONCAT

Table 7-131 Concatenate strings instruction

LAD/ FBD	Description
	CONCAT (concatenate strings) joins string parameters IN1 and IN2 to form one string provided at OUT. After concatenation, String IN1 is the left part and String IN2 is the right part of the combined string.

Supported Properties: None

Table 7-132 Data types for the parameters

Parameter	Data type	Description
IN1	String	Input string 1
IN2	String	Input string 2
OUT	String	Combined string (string 1 + string 2)

Table 7-133 ENO status

ENO	Description
True	No error
False	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range

10.2.5 INSERT

Table 7-134 Insert substring instruction

LAD/ FBD	Description
	Inserts string IN2 into string IN1. Insertion begins after the character at position P.

Supported Properties: None

Table 7-135 Data types for the parameters

Parameter	Data type	Description
IN1	String	Input string 1
IN2	String	Input string 2
P	AnyInt	Last character position in string IN1 before the insertion point for string IN2
OUT	String	Combined string (string 1 + string 2)

Table 7-136 ENO status

ENO	Description
True	No error
False	<ul style="list-style-type: none"> • P is greater than length of IN1 • The result length is greater than the max allowed string size • P is less than 0

10.2.6 DELETE

Table 7-137 Delete substring instruction

LAD/ FBD	Description
	Inserts string IN2 into string IN1. Insertion begins after the character at position P.

Supported Properties: None

Table 7-138 Data types for the parameters

Parameter	Data type	Description
IN1	String	Input string
L	AnyInt	Number of characters to be deleted
P	AnyInt	Position of the first character to be deleted: The first character of the IN string is position number 0
OUT	String	Output string

Table 7-139 ENO status

ENO	Description
True	No error
False	<ul style="list-style-type: none"> • P is greater than length of IN1 • The result length is greater than the max allowed string size • P is less than 0

10.2.7 REPLACE

Table 7-140 Replace substring instruction

LAD/ FBD	Description
	<p>Inserts string IN2 into string IN1. Insertion begins after the character at position P.</p> <p>If parameter L is equal to zero, then the string IN2 is inserted at position P of string IN1 without deleting any characters from string IN1.</p> <p>If P is equal to one, then the first L characters of string IN1 are replaced with string IN2 characters.</p>

Supported Properties: None

Table 7-141 Data types for the parameters

Parameter	Data type	Description
IN1	String	Input string
IN2	String	String of replacement characters
L	AnyInt	Number of characters to replace
P	AnyInt	Position of first character to be replaced
OUT	String	Output string

Table 7-142 ENO status

ENO	Description
True	No error
False	<ul style="list-style-type: none"> • P is greater than length of IN1 • The result length is greater than the max allowed string size • P is less than 0

10.2.8 FIND

Table 7-143 Find substring instruction

LAD/ FBD	Description
	<p>Provides the character position of the substring specified by IN2 within the string IN1. The search starts on the left. The character position of the first occurrence of IN2 string is returned at OUT. If the string IN2 is not found in the string IN1, then -1 is returned.</p>

Supported Properties: None

Table 7-144 Data types for the parameters

Parameter	Data type	Description
IN1	String	Input string 1
IN2	String	Input string 2
OUT	String	Combined string (string 1 + string 2)

Table 7-145 ENO status

ENO	Description
True	No error
False	<ul style="list-style-type: none">• P is greater than length of IN1• The result length is greater than the max allowed string size• P is less than 0

8

System Instructions

This chapter describes instructions that are not related to the IEC programming standard, but are related to CPU and its operating system hardware management. System instructions may vary from one CPU model to another due to their structural differences.

1. Memory management

1.1 RWW_NVMEM instruction

Table 8-1 Read/Write nonvolatile memory instruction

LAD/ FBD	Description
	<p>Reads tags data from permanent memory or writes (when R/W=True) them on permanent memory respect to the specified address. Writes the status on the STT and processed bytes on the CNT.</p>

Supported Properties: Inputs Count

Table 8-2 Data types for the parameters

Parameter	Data type	Description
R/W	bool	Specifies to read the data (R/W=False) or write (R/W=True)
ADDR	AnyInt	Byte address for reading or writing data
IN1...INn	Variant	Input tags 1...n
STT	Int	Operation status. -1: failed otherwise: Ok
CNT	AnyInt	The number of written or read bytes

2. System Time Management

2.1 GET_SYS_DT instruction

Table 8-3 Get CPU date and time instruction

LAD/ FBD	Description
	<p>Gets the current CPU date and time.</p>

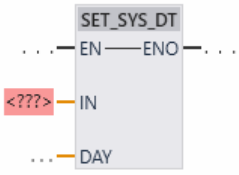
Supported Properties: None

Table 8-4 Data types for the parameters

Parameter	Data type	Description
OUT	AnyDate	Current system date and time
DAY	USINT	Day of week

2.2 SET_SYS_DT instruction

Table 8-5 Get CPU date and time instruction

LAD/ FBD	Description
	Sets the current CPU date and time.

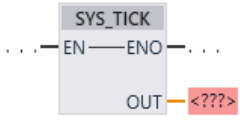
Supported Properties: None

Table 8-6 Data types for the parameters

Parameter	Data type	Description
IN	AnyDate	Desired system date and time
DAY	USINT	Day of week

2.3 SYS_TICK instruction

Table 8-7 Get CPU tick time instruction

LAD/ FBD	Description
	Gets the current CPU tick time.

Supported Properties: None

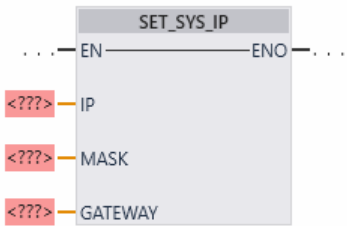
Table 8-8 Data types for the parameters

Parameter	Data type	Description
OUT	Time	Current CPU tick time

3. Comm ports management

3.1 SET_SYS_IP

Table 8-9 Set system IP instruction

LAD/ FBD	Description
	<p>Sets the IP parameters of the device by the specified parameters. Note that parameters save only when the instruction executes in startup OB.</p> <p>This instruction is applicable only for devices that have ethernet port.</p>

Supported Properties: None

Table 8-10 Data types for the parameters

Parameter	Data type	Description
IP	IP_V4	Desired system IP address
MASK	IP_V4	Desired system subnet mask
GATEWAY	IP_V4	Desired system gateway address

9

Communication Instructions

PLCs use built-in ports, such as USB, Ethernet, RS-232, RS-485, or industrial CAN to communicate with external devices (sensors, actuators) and systems (programming software, SCADA, HMI, other PLCs). Communication is carried over various industrial network protocols, like Modbus RTU, Modbus TCP, or non-protocols for raw data transmissions.

1. RS-232 interface

An RS-232 interface is rated for distances up to 15 meters (50 feet). At least three wires are required for an RS-232 interface. Wires are required for Transmit, Receive and Signal Ground. Some devices support additional wires for communication handshaking. RS-232 hardware is a full-duplex configuration, having separate Transmit and Receive lines. Each signal that transmits in an RS-232 data transmission system appears on the interface connector as a voltage with reference to a signal ground. The RS-232 receiver typically operates within the voltage range of +3 to +12 and -3 to -12 volts. The recommended cable is up to 15m (50ft) virtually any standard shielded twisted pair with drain (Belden 9502 or equivalent).

2. RS-485 interface

For multi-drop operation, drivers must be capable of tri-state operation. An RS-485 interface requires at least two wires. In a two-wire configuration, the same pair of wires is used for Transmit and Receive. The two-wire configuration utilizes half-duplex communications. Transmit driver circuits are always taken off-line or tri-stated, when not in use. This tri-state feature reduces the load on the network, allowing more devices, without the need of special hardware. This interface also uses differential drivers, supporting distances up to 1200 meters (4000 feet). In a differential system the voltage produced by the driver appears across a pair of signal lines that transmit only one signal. A differential line driver will produce a voltage from 2 to 6 volts across its A and B output terminals and will have a signal ground (C) connection. Although proper connection to the signal ground is important, it isn't used by a differential line receiver in determining the logic state of the data line. A differential line receiver senses the voltage state of the transmission line across two signal input lines, A and B. It will also have a signal ground (C) that is necessary in making the proper interface connection. If the differential input voltage V_{ab} is greater than +200 mV the receiver will have a specific logic state on its output terminal. If the input voltage is reversed to less than -200 mV the receiver will create the opposite logic state on its output terminal.

2.1 Bias resistors

RS-485 networks often require bias, or pull-up and pull-down resistors. These resistors are used to stabilize the network. By definition, in a MODBUS RTU network, it is the responsibility of the Master to provide this function. Some systems may function without these stabilizing resistors, but may be more susceptible to communication errors. Though the pull-up and pulldown resistors are the same, the value of these resistors varies from device to device.

TIP

I4 PLCs have an internal biasing circuit so, there is no need for you to bias the bus.

2.2 Termination resistors

Termination resistors are often used to reduce reflections on the network. This problem occurs most with long wires and high baud rates. Due to variations in wire and equipment, whether or not to use these terminators is usually determined by system testing. The general rule is to add them only if needed. The resistors are typically 120 ohms, and installed across the Transmit and Receive wire pairs. Normally, one resistor is installed at each end of each pair of wires.

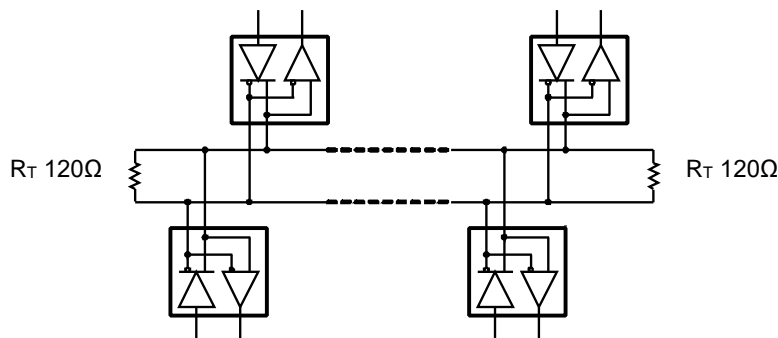


Figure 9-1 Termination of a RS-485 bus

2.3 Shielding and grounding considerations

The signal ground conductor is often overlooked when ordering cable. An extra twisted pair must be specified to have enough conductors to run a signal ground. A two-wire system then requires two twisted pairs.

It is often hard to quantify if shielded cable is required in an application or not. Since the added cost of shielded cable is usually minimal it is worth installing the first time.

2.4 Cable requirements

The type of wire to use will vary with required length. Wire with twisted pairs and an overall shield is used most often. The shield is tied to earth ground or chassis, and typically at one end only (generally at the Modbus Master side). The shield is not to be used as a signal common or ground. The recommended cable is up to 1200m (4000ft) 24 AWG twisted pair with foil shield and drain wire on each pair (Belden 9841 for 2-wire and 9729 for 4-wire or equiv.)

3. Controller Area Network (CAN) interface

The CAN communication protocol is a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP). CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. CD+AMP means that collisions are resolved through a bit-wise arbitration, based on a preprogrammed priority of each message in the identifier field of a message. The higher priority identifier always wins bus access. That is, the last logic high in the identifier keeps on transmitting because it is the highest priority. Since every node on a bus takes part in writing every bit "as it is being written," an arbitrating node knows if it placed the logic-high bit on the bus.

The ISO-11898:2003 Standard, with the standard 11-bit identifier, provides for signaling rates from 125 kbps to 1 Mbps. The standard was later amended with the "extended" 29-bit identifier. The standard 11-bit identifier field provides for 2^{11} , or 2048 different message identifiers, whereas the extended 29-bit identifier in provides for 2^{29} , or 537 million identifiers. Bus access is event-driven and takes place randomly. If two nodes try to occupy the bus simultaneously, access is implemented with a nondestructive, bit-wise arbitration. Nondestructive means that the node winning arbitration just continues on with the message, without the message being destroyed or corrupted by another node.

The allocation of priority to messages in the identifier is a feature of CAN that makes it particularly attractive for use within a real-time control environment. The lower the binary message identifier number, the higher its priority. An identifier consisting entirely of zeros is the highest priority message on a network because it holds the bus dominant the longest. Therefore, if two nodes begin to transmit simultaneously, the node that sends a last identifier bit as a zero (dominant) while the other nodes send a one (recessive) retains control of the CAN bus and goes on to complete its message. A dominant bit always overwrites a recessive bit on a CAN bus.

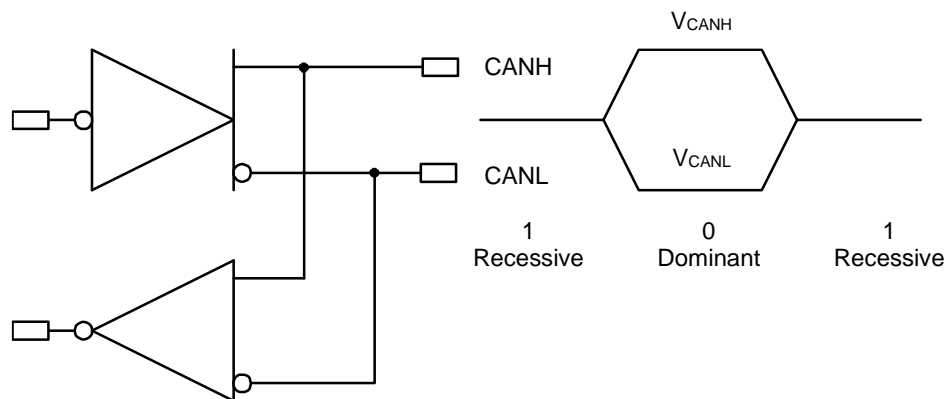


Figure 9-2 The Inverted Logic of a CAN Bus

The data rate of the bus can reach up to 1 Mbps (for 30m length cable) or up to 8 Mbps for FD (Flexible Datarate) CAN.

4. Ethernet interface

Ethernet was first developed in the 1970s and was later standardized as IEEE 802.3. Ethernet is the group of local area network (LAN) products covered by IEEE 802.3—a group of Institute of Electrical and Electronics Engineers (IEEE) standards that define the physical layer and data link layer of a wired Ethernet media access control.1 These standards also describe the rules for configuring an Ethernet network and how the elements of the network work with one another. Ethernet allows computers to connect over one network. Ethernet is the global standard for a system of wires and cables to conjoin multiple computers, devices, machines, etc., over an

organization’s single network so that all the computers can communicate with one another. Ethernet began as a single cable, making it possible for multiple devices to be connected on one network. Now, an Ethernet network can be expanded to new devices as needed. Ethernet is now the most popular and widely used network technology in the industry. With industrial Ethernet, data transmission rates range from 10 Mbps to 1 Gbps. However, 100 Mbps is the most popular speed used in industrial Ethernet applications.

While there are several Industrial Ethernet protocols to support a variety of communication requirements in the industrial automation, there are four major protocols.

4.1 Modbus TCP/IP

Modbus TCP/IP was the first Industrial Ethernet protocol introduced, and it is essentially a traditional Modbus communication that is compressed within an Ethernet transport layer protocol for transferring discrete data between control devices. It uses a simple master-slave communication where the “slave” node will not transmit data without a request from the “master” node, but it is not considered a real-time protocol.

4.2 EtherCAT

Introduced in 2003, EtherCAT is an Industrial Ethernet protocol that offer real-time communication in a master/slave configuration for automation systems. The key element of EtherCAT is the ability for all networked slaves to extract only the relevant information they need from the data packets and insert data into the frame as it transmits downstream.

4.3 Ethernet/IP

Initially released in 2000, Ethernet/IP is a widely used application-layer Industrial Ethernet protocol supported by the Open Device Vendors Association (ODVA) and supplied primarily by Rockwell Automation. It is the only Industrial Ethernet protocol that is based entirely on Ethernet standards and uses standard Ethernet physical, data link, network and transport layers. Since it uses standard Ethernet switching, it can support an unlimited number of nodes. However, it requires limited range to avoid latency and support real-time communication.

4.4 PROFINET

An application protocol developed by Siemens in conjunction with member companies of a Profibus user organization. It essentially extends Profibus I/O controller communication to Ethernet using special switches that are integrated into devices.

5. Programming instructions

5.1 Serial

5.1.1 SERIAL_INIT instruction

Table 9-1 Serial port initialize instruction

LAD/ FBD	Description
	<p>SERIAL_INIT allows you to change port parameters such as baud rate from your program.</p> <p>You can set up the initial static configuration of the port in the device configuration properties, or just use the default values. You can execute the SERIAL_INIT instruction in your program to change the configuration. Data bits will be set always to 8.</p>

Supported Properties: None

Table 9-2 Data types for the parameters

Parameter	Data type	Description
PORT	AnyInt	Desired system port identifier. It starts by 0. For other ports identifier see device technical data
BAUDRATE	AnyInt	Port baud rate. 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200
PARITY	AnyInt	Port parity. 0 = No parity, 1 = Even parity, 2 = Odd parity
STOP_BITS	AnyInt	Stop bits. 0 = No stop bit, 1 = One stop bit, 2 = Two stop bit

Table 9-3 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly

5.1.2 SERIAL_GET_STAT instruction

Table 9-4 Serial port get status instruction

LAD/ FBD	Description
	<p>SERIAL_INIT allows you to change port parameters such as baud rate from your program.</p> <p>You can set up the initial static configuration of the port in the device configuration properties, or just use the default values. You can execute the SERIAL_INIT instruction in your program to change the configuration. Data bits will be set always to 8.</p>

Supported Properties: None

Table 9-5 Data types for the parameters

Parameter	Data type	Description
PORT	AnyInt	Desired system port identifier. It starts by 0. For other ports identifier see device technical data
BUF_LVL	AnyInt	Current received bytes count in the buffer.
OVR	Bool	Overflow flag. When the serial receiving buffer overflows it means that the incoming data length is greater than the buffer size so, a part of data has been dropped
Busy	Bool	Busy flag. When the serial port is receiving data or transmitting data this flag will be True.

Table 9-6 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly

5.1.3 SERIAL_READ_BUF instruction

Table 9-7 Serial read buffer instruction

LAD/ FBD	Description
	SERIAL_READ_BUF allows you to read the input data received in serial buffer and store it in an external Byte array.

Supported Properties: None

Table 9-8 Data types for the parameters

Parameter	Data type	Description
PORT	AnyInt	Desired system port identifier. It starts by 0. For other ports identifier see device technical data
BUFFER	Byte[*]	Destination buffer for storing data
COUNT	AnyInt	The length of the data stored in the buffer

Table 9-9 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly

5.1.4 SERIAL_SEND_BUF instruction

Table 9-10 Serial send buffer instruction

LAD/ FBD	Description
	SERIAL_SEND_BUF allows you to transmit the input data stored in Byte array buffer on the serial port.

Supported Properties: None

Table 9-11 Data types for the parameters

Parameter	Data type	Description
PORT	AnyInt	Desired system port identifier. It starts by 0. For other ports identifier see device technical data
ADDR	AnyInt	The start position on the buffer to be sent
COUNT	AnyInt	The length of data to be sent
BUFFER	Byte[*]	Source buffer containing data

Table 9-12 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly or the port is busy

6. Modbus communication

6.1 Overview of Modbus RTU and TCP communication

6.1.1 Modbus function codes

- A CPU operating as a Modbus RTU master (or Modbus TCP client) can read/write both data and I/O states in a remote Modbus RTU slave (or Modbus TCP server). Remote data can be read and processed in the user program.
- A CPU operating as a Modbus RTU slave (or Modbus TCP server) allows a supervisory device to read/write both data and I/O states in a remote CPU. The supervisor device can write new values in remote CPU memory that can be processed in the user program.

Table 9-13 Read data functions: Read remote I/O and program data

Modbus function code	Read slave (server) functions - standard addressing
01	Read output bits: 1 to 2000 bits per request
02	Read input bits: 1 to 2000 bits per request
03	Read Holding registers: 1 to 125 words per request
04	Read input words: 1 to 125 words per request

Table 9-14 Write data functions: Write remote I/O and modify program data

Modbus function code	Write slave (server) functions - standard addressing
05	Write one output bit: 1 bit per request
06	Write one holding register: 1 word per request
16	Write one or more holding registers: 1 to 123 words per request

- Modbus ID 0 broadcasts a message to all slaves (with no slave response). The broadcast function is not available for Modbus TCP, because communication is connection based.

Table 9-15 Modbus network station addresses

Station	Address
RTU station	1 to 247
TCP station	IP address and port number

6.1.2 Modbus memory addresses

The actual number of Modbus memory addresses available depends on the CPU model, how much application memory exists, and how much CPU memory is used by other program data. The table below gives the nominal value of the address range.

Table 9-16 Modbus memory addresses

Station	Address range
RTU station	1K
TCP station	1K

6.1.3 Modbus RTU communication

Modbus RTU (Remote Terminal Unit) is a standard network communication protocol that uses the RS485 electrical connection for serial data transfer between Modbus network devices.

Modbus RTU uses a master/slave network where all communications are initiated by a single Master device and slaves can only respond to a master's request. The master sends a request to one slave address and only that slave address responds to the command.

6.1.4 Modbus TCP communication

Modbus TCP (Transmission Control Protocol) is a standard network communication protocol that uses the Ethernet connector on the CPU for TCP/IP communication. No additional communication hardware module is required.

Modbus TCP uses Open User Communications (OUC) connections as a Modbus communication path. Multiple client-server connections may exist. Mixed client and server connections are supported up to the maximum number of connections allowed by the CPU model (see technical data).

A Modbus TCP client (master) must control the client-server connection with the DISCONN parameter. The basic Modbus client actions are shown below.

Initiate a connection to a particular server (slave) IP address and IP port number

Initiate client transmission of a Modbus messages and receive the server responses

When desired, initiate the disconnection of client and server to enable connection with a different server.

TIP

You can change the default port number for Modbus TCP server in Online & Diagnostic/Options.

6.1.5 Modbus RTU instructions in your program

- MB_MASTER: The Modbus master instruction enables the CPU to act as a Modbus RTU master device and communicate with one or more Modbus slave devices.
- MB_SLAVE: The Modbus slave instruction enables the CPU to act as a Modbus RTU slave device and communicate with a Modbus master device.

6.1.6 Modbus TCP instructions in your program

- MB_CLIENT: Make client-server TCP connection, send command message, receive response, and control the disconnection from the server.
- MB_SERVER: Connect to a Modbus TCP client upon request, receive Modbus message, and send response.

6.2 Modbus RTU

6.2.1 MB_SLAVE

LAD/ FBD	Description
	<p>The MB_SLAVE instruction allows your program to communicate as a Modbus slave through a RS485. When a remote Modbus RTU master issues a request, your user program responds to the request by MB_SLAVE execution. Your program must config the port by an execution of the SERIAL_INIT instruction before start the MB_SLAVE.</p>

Supported Properties: None

Table 9-17 Data types for the parameters

Parameter	Data type	Description
PORT	AnyInt	Desired system port identifier. It starts by 0. For other ports identifier see device technical data
SLAVE_ID	AnyInt	The station address of the Modbus slave in standard addressing range (1 to 247)
REG_ADDR	AnyInt	Pointer to the Modbus Holding Register in the bit memory (M). Specifies the starting register address (word address) of the data to be accessed by other masters in M memory.
REG_COUNT	AnyInt	Holding Registers data Length: Specifies the number of words to be accessed in by other masters in M memory.
STOP	Bool	Starts or stops the MB_SLAVE functions. <ul style="list-style-type: none"> • Null or False: Start • True: Stop

Table 9-18 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly: <ul style="list-style-type: none"> • The serial port is not initialized • PORT is out of range • SLAVE_ID = 0 • REG_ADDR + REG_COUNT is greater than the allowed memory area size • REG_COUNT is greater than 1024

Table 9-19 Supported function codes and mapping of Modbus addresses to the

Function code	Address range	Bit memory (M) range	Operation and data
3	0 to REG_COUNT-1	REG_ADDR to REG_COUNT-1	Read Holding registers
6			Write one holding register
16			Write multiple holding registers

The following table shows examples of Modbus address to holding register mapping that is used for Modbus function codes 03 (read words), 06 (write word), and 16 (write words). The actual upper limit of bit memory (M) address is determined by the maximum application memory limit and M memory limit, for each CPU model.

Table 9-20 Mapping of Modbus addresses to CPU memory

Modbus Master Address	Word address	Example for M address when REG_ADDR=16
0	REG_ADDR	%MW32
1	REG_ADDR+1	%MW34
2	REG_ADDR+2	%MW36
3	REG_ADDR+3	%MW38
4	REG_ADDR+4	%MW40
5	REG_ADDR+5	%MW42

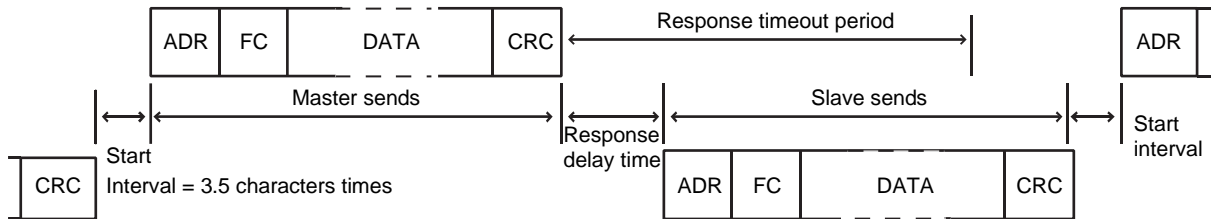
Modbus slave communication rules

- SERIAL_INIT must be executed to configure a port, before a MB_SLAVE instruction can communicate through that port.
- If a port is to respond as a slave to a Modbus master, then do not program that port with the MB_MASTER instruction.

- Only one instruction call of MB_SLAVE can be used with a given port, otherwise erratic behavior may occur.
- The MB_SLAVE instruction must execute only one time in your program. Then its internal functions will be executed periodically at a rate that allows it to make a timely response to incoming requests from a Modbus master. It is recommended that you execute MB_SLAVE in a Startup OB or in other OBs only one time at the CPU startup. Executing MB_SLAVE from a cyclic interrupt OB is not possible.

Modbus signal timing

Modbus slave functions will be executed periodically after calling MB_SLAVE instruction to receive each request from the Modbus master and then respond as required. The frequency of execution for MB_SLAVE is dependent upon the response timeout period of the Modbus master. This is illustrated in the following diagram.

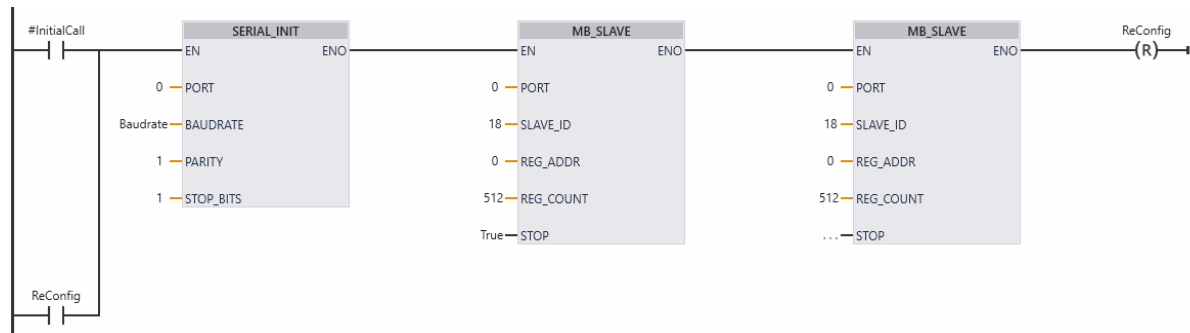


The response timeout period is the amount of time a Modbus master waits for the start of a response from a Modbus slave. This time period is not defined by the Modbus protocol, but is a parameter of each Modbus master. The frequency of execution (the time between one execution and the next execution) of MB_SLAVE must be based on the particular parameters of your Modbus master.

6.2.2 Modbus RTU slave example program

SERIAL_INIT shown below initializes the RS485 port parameters at the CPU startup by the first scan flag or each time they are changed by an HMI device (by the "Reconfig" flag). Then, the Modbus slave stops and starts again in order to new configuration take effect.

The Modbus holding register is configured for 512 words starting at %MW0.



6.2.3 MB_MASTER

Table 9-21 MB_MASTER instruction

LAD/ FBD	Description
	<p>The MB_MASTER instruction communicates as a Modbus master using a port that was configured by a previous execution of the SERIAL_INIT instruction.</p>

Supported Properties: None

Table 9-22 Data types for the parameters

Parameter	Data type	Description
PORT	AnyInt	Desired system port identifier. It starts by 0. For other ports identifier see device technical data
ID	AnyInt	Modbus RTU station address. The value of 0 is reserved for broadcasting a message to all Modbus slaves. Modbus function codes 05, 06, 15 and 16 are the only function codes supported for broadcast.
FC	AnyInt	Function code.
REG_ADDR	AnyInt	Starting Address in the slave: Specifies the starting register address (word address) of the data to be accessed in the Modbus slave.
REG_COUNT	AnyInt	Data Length: Specifies the number of bits or words to be accessed in this request. See the Modbus functions table below for valid lengths.
R/W32	Bool	False = Normal Modbus operation True = Concatenates two consecutive registers. When the R/W32 is True the BUFFER must be DWord array.
TIMEOUT	AnyInt	Amount of time (in milliseconds) to wait for a blocked Modbus Master instruction before removing this instruction as being ACTIVE. This can occur, for example, when a Master request has been issued and then the program stops calling the Master function before it has completely finished the request. The time value must be greater than 0 and less than 65535 milliseconds, or an error occurs.
BUFFER	AnyBit[*]	Source buffer containing data. When you access data by 01,02 and 05 the BUFFER data type must be an array of Bool, otherwise an array of Word.
STAT	AnyInt	Execution condition code
ERROR	AnyInt	The ERROR bit is True for one scan, after the MB_CLIENT execution was terminated with an error. The error code value at the STAT parameter is valid only during the single cycle where ERROR = True.
Busy	Bool	<ul style="list-style-type: none"> False = No MB_CLIENT operation in progress True = MB_CLIENT operation in progress

Table 9-23 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly <ul style="list-style-type: none"> • Invalid value for TIMEOUT • Invalid datatype for BUFFER

Modbus master communication rules

- SERIAL_INIT must be executed to configure a port before a MB_MASTER instruction can communicate with that port.
- If a port is to be used to initiate Modbus master requests, that port should not be used by MB_SLAVE . One or more MB_MASTER execution can be used with that port.
- The Modbus instructions do not use communication interrupt events to control the communication process. Your program must poll the MB_MASTER instruction for transmit and receive complete conditions.
- It is recommended that you call all MB_MASTER execution for a given port from a loop executor OB. Modbus master instructions may execute in only one of the program cycle or cyclic/time of day execution levels. They must not execute in both execution priority levels. Pre-emption of a Modbus Master instruction by another Modbus master instruction in a higher priority execution priority level will result in improper operation. Modbus master instructions must not execute in the startup or stop OBs.
- Once a master instruction initiates a transmission, this instruction must be continually executed with the EN input enabled until a BUSY=False state or ERROR<>0 state is returned. A particular MB_MASTER is considered active until one of these two events occurs. While an instruction is active, any call to any other instruction will result in an error. If the continuous execution of the original instruction stops, the request state remains active for a period of time specified by the TIMEOUT input. Once this period of time expires, the next master instruction called will become the active instance. This prevents a single Modbus master instance from monopolizing or locking access to a port. If the original active instruction is not enabled then the next instruction will be executed.

Modbus function codes

The MB_MASTER instruction uses a Function Code input (FC) determine the Function Code that is used in the actual Modbus message. The following table shows the eligible Modbus function codes.

Table 9-24 Modbus functions

Function code	Data length	Operation and data
3	1 to 123	Read Holding registers
6	1	Write one holding register
16	1 to 123	Write multiple holding registers

Status codes

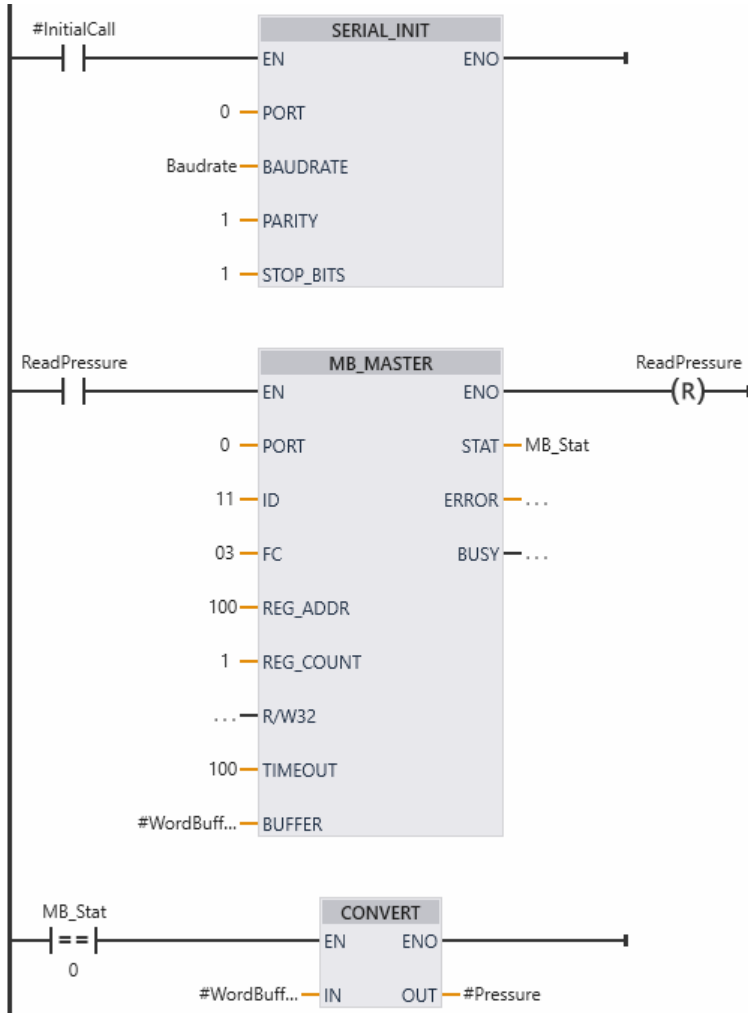
Table 9-25 MB_MASTER execution condition codes (communication and configuration errors)

STATUS	Description
0	No error
255	Slave timeout. Check baud rate, parity, and wiring of slave. Invalid Modbus station address
18	Invalid function code
16	Invalid port ID value or error with INIT_SERIAL instruction
129	Invalid Data Address value
17	Invalid Data Length value
254	The message was terminated as a result of the specified length exceeding the total buffer size.
130	Data value error

6.2.4 Modbus RTU master example program

RS-485 port is initialized only once during start-up by using SERIAL_INIT triggered by the first scan flag. Execution of SERIAL_INIT in this manner should only be done when the serial port configuration will not change at runtime.

One MB_MASTER instruction is used in the cyclic program OB to communicate with a single slave. Additional MB_MASTER instructions can be used in the OB to communicate with other slaves, or one MB_MASTER instruction could be re-used to communicate with additional slaves.



6.3 Modbus TCP

6.3.1 MB_SERVER

Table 9-26 MB_SERVER instruction

LAD/ FBD	Description
	<p>MB_SERVER communicates as a Modbus TCP server through the Ethernet connector on the CPU (all CPU models may not must support). No additional communication hardware module is required. MB_SERVER can accept a request to connect with Modbus TCP client, receive a Modbus function request, and send a response message.</p>

Supported Properties: None

Table 9-27 Data types for the parameters

Parameter	Data type	Description
IP_LIST	IP_V4[*]	An array of IP_V4 structure for eligible clients that connect this server. All requests from other clients that are not mentioned in this array will be ignored. The maximum length of the array is 4.
REG_ADDR	AnyInt	Pointer to the Modbus Holding Register in the bit memory (M). Specifies the starting register address (word address) of the data to be accessed by other masters in M memory.
REG_COUNT	AnyInt	Holding Registers data Length: Specifies the number of words to be accessed in by other masters in M memory.
STOP	Bool	Starts or stops the MB_SERVER functions. <ul style="list-style-type: none"> • Null or False: Start • True: Stop This allows your program to control when a connection is accepted. Whenever this input is enabled, no other operation will be attempted.

Table 9-28 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly <ul style="list-style-type: none"> • REG_ADDR + REG_COUNT is greater than the allowed memory area size • REG_COUNT is greater than 1024 • IP_V4 array length is greater than 4

Table 9-29 Supported function codes and mapping of Modbus addresses to the

Function code	Address range	Bit memory (M) range	Operation and data
03	0 to REG_COUNT-1	REG_ADDR to REG_COUNT-1	Read Holding registers
06			Write one holding register
16			Write multiple holding registers

The following table shows examples of Modbus address to holding register mapping that is used for Modbus function codes 03 (read words), 06 (write word), and 16 (write words). The actual upper limit of bit memory (M) address is determined by the maximum application memory limit and M memory limit, for each CPU model.

Table 9-30 Mapping of Modbus addresses to CPU memory

Modbus Client Address	Word address	Example for M address when REG_ADDR=16
0	REG_ADDR	%MW32
1	REG_ADDR+1	%MW34
2	REG_ADDR+2	%MW36
3	REG_ADDR+3	%MW38
4	REG_ADDR+4	%MW40
5	REG_ADDR+5	%MW42

 TIP

Multiple server connections may be created. This permits a single PLC to establish concurrent connections to multiple Modbus TCP clients. The maximum number of Open User Communications connections allowed by the PLC is 4.

TIP

The default MB_SERVER port is 502. If you want to change the port value, go to “Online & Diagnostic”/Options.

6.3.2 MB_SERVER example

The Modbus holding register is configured for 1024 words starting at %MW0.

The IPList is an array with two elements of IP_V4 structure.



Two clients (192.168.1.32 and 192.168.1.33) will be allowed to transfer data to the server.

6.3.3 MB_CLIENT

Table 9-31 MB_CLIENT instruction

LAD/ FBD	Description
	<p>MB_CLIENT communicates as a Modbus TCP client through the Ethernet connector on the CPU. No additional communication hardware module is required.</p> <p>MB_CLIENT can make a client-server connection, send a Modbus function request, receive a response, and control the disconnection from a Modbus TCP server.</p>

Supported Properties: None

Table 9-32 Data types for the parameters

Parameter	Data type	Description
REQ	Bool	False = No Modbus communication request True = Request to communicate with a Modbus TCP server
R/W	Bool	Specifies condition whether write data to server (Function code 16) or read data from server (Function code 03).
REG_ADDR	AnyInt	Starting Address in the slave: Specifies the starting register address (word address) of the data to be accessed in the Modbus slave.

REG_COUNT	AnyInt	Data Length: Specifies the number of bits or words to be accessed in this request. See the Modbus functions table below for valid lengths.
IP	IP_V4	Modbus TCP server IP address.32-bit IPv4 IP address of the Modbus TCP server to which the client will connect and communicate using the Modbus TCP protocol.
ID	AnyInt	ID for Modbus TCP server socket. You can communicate concurrently up to 2 Modbus TCP server. 0 or 1
TIMEOUT	AnyInt	Amount of time (in milliseconds) to wait for a blocked Modbus Client instruction before removing this instruction as being ACTIVE. This can occur, for example, when a client request has been issued and then the program stops calling the client function before it has completely finished the request.
DISCONN	Bool	The DISCONN parameter allows your program to control connection and disconnection with a Modbus server device. If DISCONN <> True and a connection does not exist, then MB_CLIENT attempts to make a connection to the assigned IP address and port number. If DISCONNECT = True and a connection exists, then a disconnect operation is attempted. Whenever this input is enabled, no other operation will be attempted.
BUFFER	Word[*]	Source buffer containing data.
STAT	AnyInt	Execution condition code.
ERROR	Bool	The ERROR bit is True for one scan, after the MB_CLIENT execution was terminated with an error. The error code value at the STAT parameter is valid only during the single cycle where ERROR = True.
Busy	Bool	<ul style="list-style-type: none"> • False = No MB_CLIENT operation in progress • True = MB_CLIENT operation in progress

Table 9-33 ENO status

ENO	Description
True	No error
False	One or more configs is not specified correctly

REQ parameter

False = No Modbus communication request True = Request to communicate with a Modbus TCP server If no instruction of MB_CLIENT is active and parameter DISCONN=False, when REQ=True a new Modbus request will start. If the connection is not already established then a new connection will be made.

As soon as the current request is completed, a new request can be processed if MB_CLIENT is executed with REQ=True.

BUFFER parameter

Assigns a buffer to store data read/written to/from a Modbus TCP server. The data buffer must be an array of Word in local or global memory.

Status codes

Table 9-34 MB_MASTER execution condition codes (communication and configuration errors)

STATUS	Description
0	Disconnected from server
256	Connecting to server
257	Connected to server
258	Server timeout. Check server IP address, port and wiring
21	The message was terminated as a result of the specified length exceeding the total buffer size.
22	Server error
23	Invalid Data Address value, Invalid Data Length value

Only 1 client can be active at any given time. Once a client completes its execution, the next client begins execution. Your program is responsible for the order of execution.

Modbus client requests can be sent over different connections. To accomplish this, different IP addresses, and connection IDs must be used.

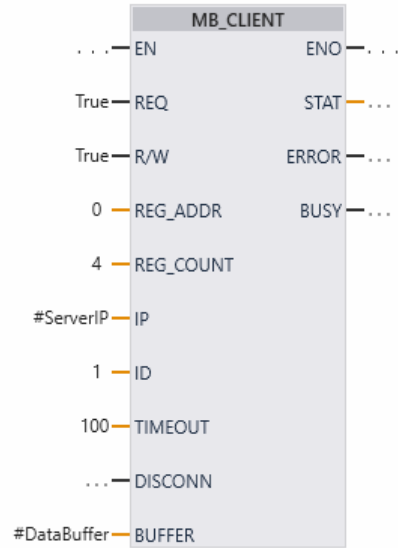
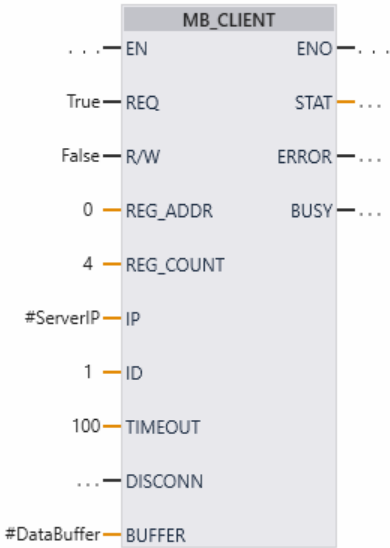
NOTICE

You can connect to a maximum of two different servers by MB_CLIENT instructions simultaneously.

6.3.4 MB_CLIENT example

Read holding register words.

Request to write on server holding registers.



10

IEC 61131-3 Solutions

This chapter provides IEC 61131-3 examples implementation that you can apply in your projects based on your needs. To see the source of these solutions you can see ANNEX F in Second edition of IEC 61131-3 standard documentation. All of these solutions are pre-compiled function blocks written by FBD language. By placing each of them on a network its source FB will be included in system program blocks.

1. CMD_MONITOR instruction

Table 10-1 CMD_MONITOR FB instruction

LAD/ FBD	Description
	<p>Example function block CMD_MONITOR illustrates the control of an operative unit which is capable of responding to a Boolean command (the CMD output) and returning a Boolean feedback signal (the FDBK input) indicating successful completion of the commanded action. The function block provides for manual control via the MAN_CMD input, or automated control via the AUTO_CMD input, depending on the state of the AUTO_MODE input (0 or 1 respectively). Verification of the MAN_CMD input is provided via the MAN_CMD_CHK input, which must be 0 in order to enable the MAN_CMD input.</p> <p>If confirmation of command completion is not received on the FDBK input within a predetermined time specified by the T_CMD_MAX input, the command is cancelled and an alarm condition is signaled via the ALRM output. The alarm condition may be cancelled by the ACK (acknowledge) input, enabling further operation of the command cycle.</p>

Supported Properties: None

Table 10-2 Data types for the parameters

Parameter	Data type	Description
AUTO_CMD	Bool	Automated command
AUTO_MODE	Bool	AUTO_CMD enable
MAN_CMD	Bool	Manual Command
MAN_CMD_CHK	Bool	Negated MAN_CMD to debounce
T_CMD_MAX	Time	Max time from CMD to FDBK
FDBK	Bool	Confirmation of CMD completion by operative unit
ACK	Bool	Acknowledge/cancel ALRM
CMD	Bool	Command to operative unit
ALRM	Bool	T_CMD_MAX expired without FDBK

2. STACK_INT FB instruction

Table 10-3 STACK_INT FB instruction

LAD/ FBD	Description
	<p>This function block provides a stack of up to 128 integers. The usual stack operations of PUSH and POP are provided by edge-triggered Boolean inputs. An overriding reset (R1) input is provided; the maximum stack depth (N) is determined at the time of resetting. In addition to the top-of-stack data (OUT), Boolean outputs are provided indicating stack empty and stack overflow states.</p>

Supported Properties: None

Table 10-4 Data types for the parameters

Parameter	Data type	Description
PUSH	Bool	Basic stack operations
POP	Bool	Basic stack operations
R1	Bool	Over-riding reset
XIN	Int	Input to be pushed
N	Int	Maximum depth after reset
EMPTY	Bool	Stack empty
OFLO	Bool	Stack overflow
OUT	Int	Top of stack data

3. LAG1 FB instruction

Table 10-5 LAG1 FB instruction

LAD/ FBD	Description
	This function block implements a first-order lag filter.

Supported Properties: None

Table 10-6 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	1 = run, 0 = reset
XIN	Real	Input variable
TAU	Time	Filter time constant
CYCLE	Time	Sampling time interval
XOUT	Real	Filtered output

4. DELAY FB instruction

Table 10-7 DELAY FB instruction

LAD/ FBD	Description
	This function block implements an N-sample delay.

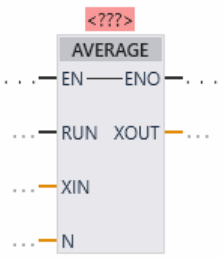
Supported Properties: None

Table 10-8 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	1 = run, 0 = reset
XIN	Real	Input variable
N	Int	$0 \leq N < 12$
XOUT	Real	Delayed output

5. AVERAGE FB instruction

Table 10-9 AVERAGE FB instruction

LAD/ FBD	Description
	This function block implements a running average over N samples.

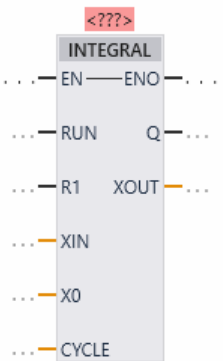
Supported Properties: None

Table 10-10 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	1 = run, 0 = reset
XIN	Real	Input variable
N	Int	$0 \leq N < 12$
XOUT	Real	Averaged output

6. INTEGRAL FB instruction

Table 10-11 INTEGRAL FB instruction

LAD/ FBD	Description
	This function block implements integration over time.

Supported Properties: None

Table 10-12 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	1 = integrate, 0 = hold

R1	Bool	Overriding reset
XIN	Real	Input variable
X0	Real	Initial value
CYCLE	Time	Sampling period
Q	Bool	NOT R1
XOUT	Real	Integrated output

7. DERIVATIVE FB instruction

Table 10-13 DERIVATIVE FB instruction

LAD/ FBD	Description
	This function block implements differentiation with respect to time.

Supported Properties: None

Table 10-14 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	0 = reset
XIN	Real	Input to be differentiated
CYCLE	Time	Sampling period
XOUT	Real	Differentiated output

8. HYSTERESIS FB instruction

Table 10-15 HYSTERESIS FB instruction

LAD/ FBD	Description
	This function block implements Boolean hysteresis on the difference of REAL inputs.

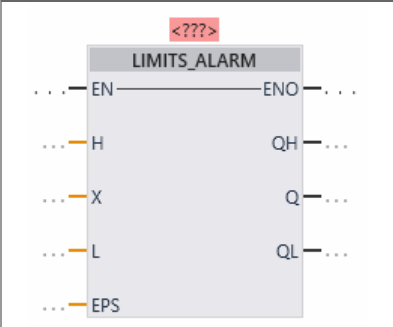
Supported Properties: None

Table 10-16 Data types for the parameters

Parameter	Data type	Description
XIN1	Real	Input 1
XIN2	Real	Input 2
EPS	Real	Epsilon
Q	Bool	$XIN1 > XIN2 + EPS = 1$, $XIN1 - EPS < XIN2 = 1$

9. LIMITS_ALARM FB instruction

Table 10-17 LIMITS_ALARM FB instruction

LAD/ FBD	Description
	This function block implements a high/low limit alarm with hysteresis on both outputs.

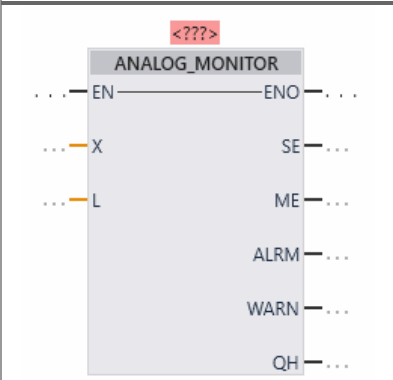
Supported Properties: None

Table 10-18 Data types for the parameters

Parameter	Data type	Description
H	Real	High limit
X	Real	Variable value
L	Real	Lower limit
EPS	Real	Hysteresis
QH	Bool	High flag
Q	Bool	Alarm output
QL	Bool	Low flag

10. ANALOG_MONITOR FB instruction

Table 10-19 ANALOG_MONITOR FB instruction

LAD/ FBD	Description
	This function block implements analog signal monitoring.

Supported Properties: None

Table 10-20 Data types for the parameters

Parameter	Data type	Description
X	Real	Variable value
L	ANALOG_LIMITS	Analog monitoring parameters structure
SE	Bool	Signal error
ME	Bool	Measurement error

ALRM	Bool	Alarm
WARN	Bool	Warning
QH	Bool	1 = Signal high

11. IEC_PID FB instruction

Table 10-21 IEC_PID FB instruction

LAD/ FBD	Description
	<p>This function block implements Proportional + Integral + Derivative control action. The functionality is derived by functional composition of previously declared function blocks.</p>

Supported Properties: None

Table 10-22 Data types for the parameters

Parameter	Data type	Description
AUTO	Bool	0 - manual, 1 - automatic
PV	Real	Process variable
SP	Real	Set point
X0	Real	Manual output adjustment typically from transfer station
KP	Real	Proportionality constant
TR	Real	Reset time
TD	Real	Derivative time constant
CYCLE	Time	Sampling period
XOUT	Real	Control signal

12. RAMP FB instruction

Table 10-23 RAMP FB instruction

LAD/ FBD	Description
	This function block implements a time-based ramp.

Supported Properties: None

Table 10-24 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	0 - track X0, 1 - ramp to/track X1
X0	Real	Start value
X1	Real	Target value
TR	Real	Ramp duration
CYCLE	Time	Sampling period
BUSY	Bool	BUSY = 1 during ramping period
XOUT	Real	Output value

13. TRANSFER FB instruction

Table 10-25 TRANSFER FB instruction

LAD/ FBD	Description
	This function block implements a manual transfer station with bump less transfer.

Supported Properties: None

Table 10-26 Data types for the parameters

Parameter	Data type	Description
AUTO	Bool	1 - track X0, 0 - ramp or hold
XIN	Real	Typically, from PID Function Block
FAST_RATE	Real	Up ramp slopes
SLOW_RATE	Real	Down ramp slopes
FAST_UP	Bool	Typically pushbuttons
SLOW_UP	Bool	
FAST_DOWN	Bool	
SLOW_DOWN	Bool	
CYCLE	Time	Sampling period
XOUT	Real	Output value

11

Monitor and Control Instructions

This chapter will help you when selecting, configuring, and assigning parameters to a controller block for your control task. It introduces you to the functions of the configuration tool and explains how you use it.

To understand this chapter, you should be familiar with automation and process control engineering concepts.

1. Designing Digital Controllers

1.1 Process Characteristics and Control

1.1.1 Process Characteristics and the Controller

The static behavior (gain) and the dynamic characteristics (time lag, dead time, reset times etc.) of the process to be controlled have a significant influence on the type and time response of the signal processing in the controller responsible for keeping the process stable or changing the process according to a selected time schedule.

The process has a special significance among the components of the control loop. Its characteristics are fixed either by physical laws or by the machinery being used and can hardly be influenced. A good control result is therefore only possible by selecting the controller type best suited to the particular process and by adapting the controller to the time response of the process.

Precise knowledge of the type and characteristic data of the process to be controlled is indispensable for structuring and designing the controller and for selecting the dimensions of its static (P mode) and dynamic (I and D modes) parameters.

1.1.2 Process Analysis

To design the controller, you require exact data from the process that you obtain by means of a transfer function following a step change in the setpoint. The (graphical) analysis of this (time) function allows you to draw conclusions about the selection of the most suitable controller function and the dimensions of the controller parameters to be set.

Before describing the use of the Configuration Standard PID Control tool the next sections briefly look at the most common processes involved in automation. You may possibly require this information to help you to decide the best procedure for the analysis and simulation of the process characteristics.

1.1.3 Type and Characteristics of the Process

The following processes will be analyzed in greater detail:

- Self-regulating process
- Self-regulating process with dead time
- Process with integral action

Self-regulating Process

Most processes are self-regulating, in other words, after a step change in the manipulated variable, the process (controlled) variable approaches a new steady-state value. The time response of the system can therefore be determined by plotting the curve of the process variable with respect to time $PV(t)$ after a step change in the manipulated variable MV by a value greater than 1.5% of its total range.

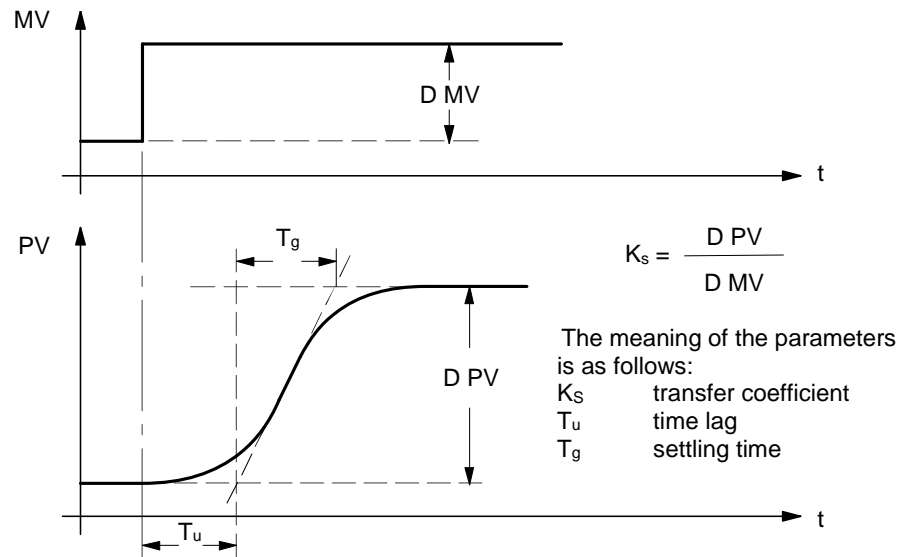


Figure 11-1 Step Response of a Self-Regulating Process (first order)

If the process response within the manipulated variable range is linear, the transfer coefficient K_s indicates the gain of the control loop. From the ratio of the time lag to the settling time T_u/T_g , the controllability of the process can be estimated. The smaller this value is, in other words the smaller the time lag relative to the settling time, the better the process can be controlled.

According to the values T_u and T_g , the time response of a process can be roughly classified as follows:

$T_u < 0.2$ min and $T_g < 2$ min → fast process

$T_u > 0.5$ min and $T_g > 5$ min → slow process

The absolute value of the settling time therefore has a direct influence on the sampling time of the controller: The higher T_g is, in other words the slower the process reaction, the higher the sampling time that can be selected.

Self-Regulating Process with Dead Time

Many processes involving transportation of materials or energy (pipes, conveyor belts etc.) have a time response similar to that shown in previous figure. This includes a start-up time T_d made up of the actual dead time and the time lag of the self-regulating process. In terms of controllability of the process it is extremely important that T_d remains small relative to T_g or in other words that the relationship $T_d/T_g \leq 1$ is maintained.

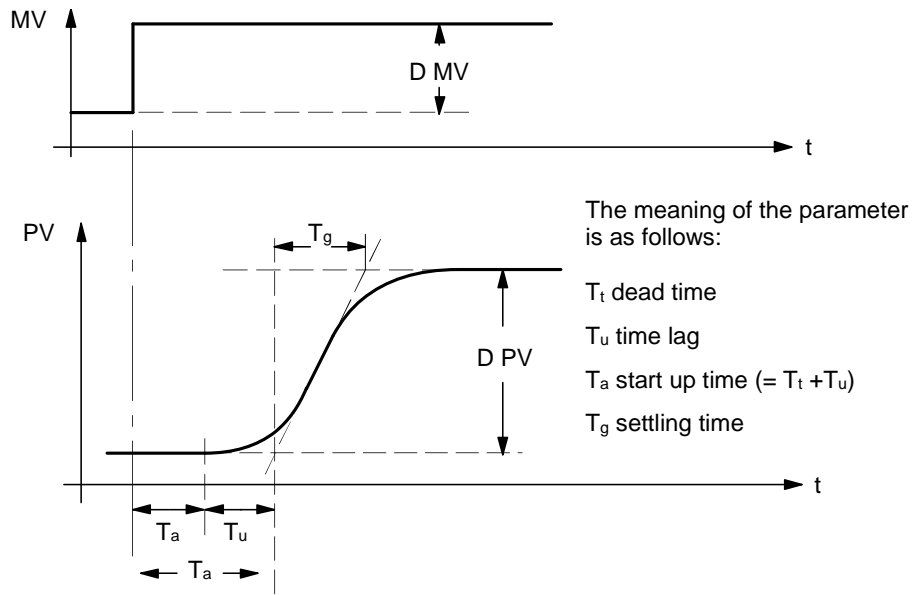


Figure 11-2 Step Response of a Self-Regulating Process with Dead Time (T_1 -PT Process)

Since the controller does not receive any signal change from the transmitter during the dead time, its interventions are obviously delayed and the control quality is therefore reduced. When using a standard controller, such effects can be partly eliminated by choosing a new location for the measuring sensor.

Process with Integral Action

Here, the slope of the ramp of the process variable (PV) after changing the manipulated variable by a fixed amount is inversely proportional to the value of the integration time constant (reset time) TI.

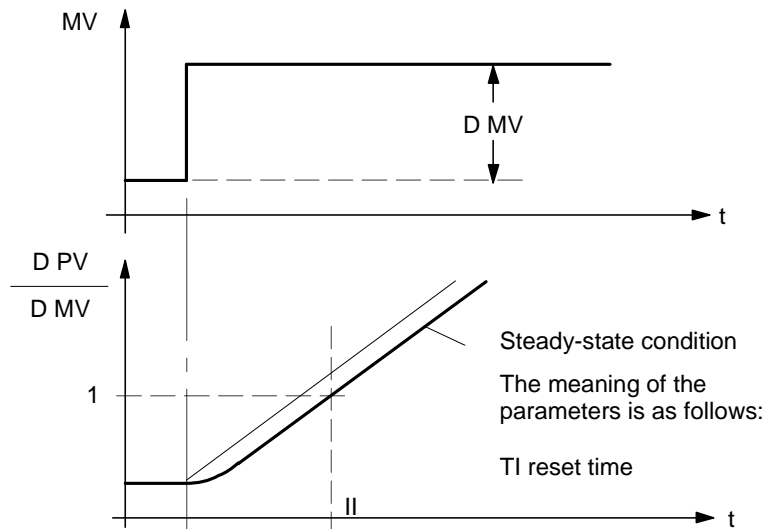


Figure 11-3 Step Response of a Non Self-Regulating Process (I Process)

Processes with an I component are, for example liquid level processes in which the level can be raised or lowered at different rates depending on the opening of the final control element. Important processes involving the I action are also the commonly used motor drives with which the rate of change of a traversing movement is directly proportional to the speed of the drive.

If no disturbance variables occur before the I element of a process with integral action (which is usually the case), a controller without I action should be used. The effects of a disturbance variable at the process input can usually be eliminated by feedforward control without using an I action in the controller.

1.2 Feedforward Control

Disturbance variables affecting the process must be compensated by the controller. Constant disturbance variables are compensated by controllers with an I action. The control quality is not affected.

Dynamic disturbance variables, on the other hand, have a much greater influence on the quality of the control. Depending on the point at which the disturbance affects the control loop and the time constants of sections of the loop after the disturbance, error signals of differing size and duration occur that can only be eliminated by the I action in the controller.

This effect can be avoided in situations where the disturbance variable "measuring" can be measured. By feeding the measured disturbance variable forward to the output of the controller, the disturbance variable can be compensated and the controller reacts much faster to the disturbance variable.

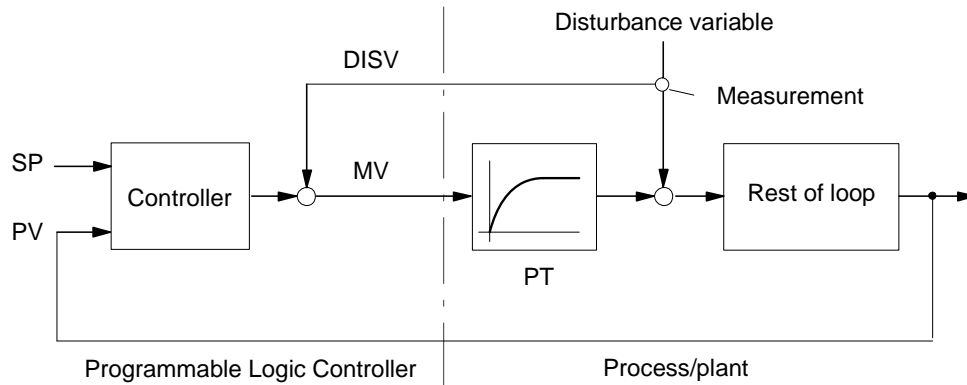


Figure 11-4 Compensating a Disturbance Affecting Process Input (Signal Names of the Standard PID Control)

1.3 Multi-Loop Controls

1.3.1 Processes with Inter-dependent Process Variables

The Standard Controller product contains prepared examples with which you can implement multi-loop controls quickly and easily. Using such control structures always has advantages when dealing with processes that have interdependent process variables.

The next sections describe the design of these controller structures and how they can be used.

Multi-loop Ratio Controls

Whenever the relationship between two or more process variables in a process is more important than keeping its absolute values constant, ratio control is necessary.

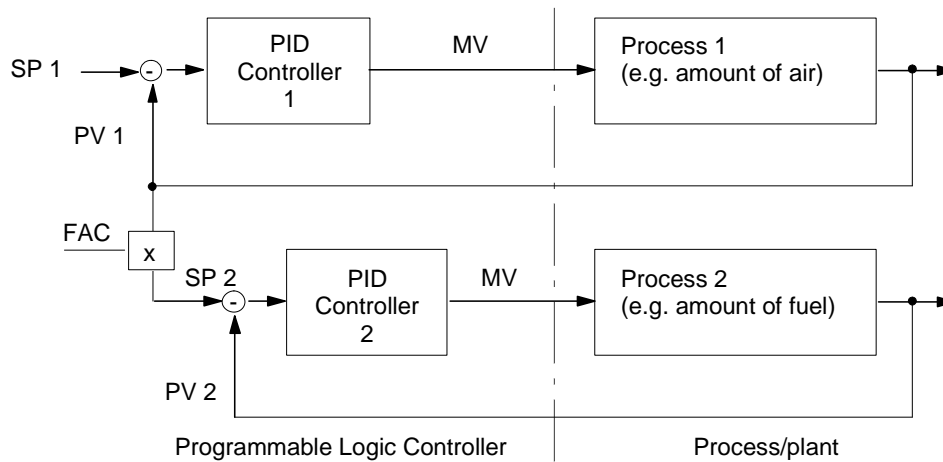


Figure 11-5 Ratio Control with Two Loops

Generally, the process variables that must be maintained in a preset ratio involve flow rates or volumes as found in combustion processes. In above figure, the amount of fuel in control loop 2 is controlled in a ratio selected with **FAC** to the amount of air set at **SP1**.

Blending Control

In a blending process, both the total amount of materials to be mixed and the ratio of the components making up the total product must be kept constant.

Based on the principle of ratio control, these requirements result in a control structure in which the amount of each component of the mixture must be controlled. The setpoints of the components are influenced by the fixed proportion or ratio factors (FAC) and by the manipulated variable of the controller responsible for the total amount (Following figure).

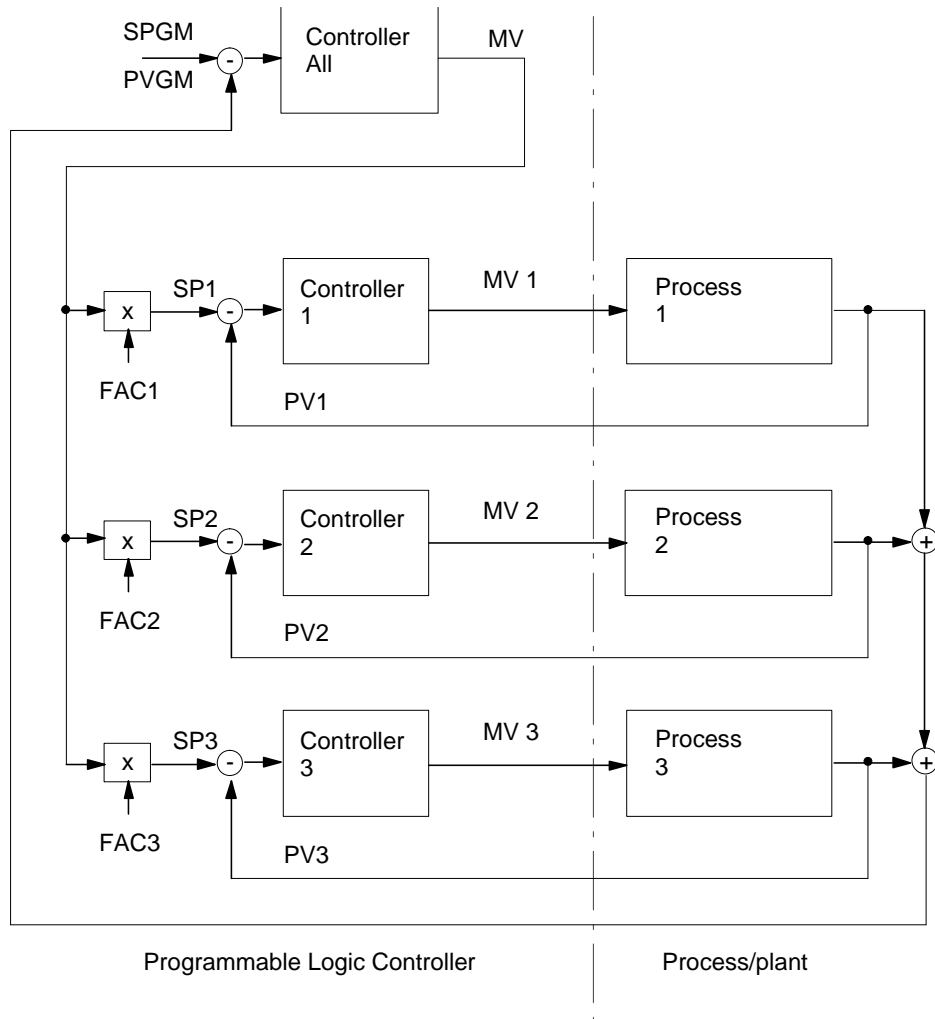


Figure 11-6 Blending control for three components

The controller structure for the blending control contains a controller with a continuous output for controlling the total amount ALL and three controllers for the secondary control loops of the individual components 1 to 3, that make up the total amount according to the factors FAC1 to FAC3 (addition).

Cascade Control

If a process includes not only the actual process variable to be controlled but also a secondary process variable that can be controlled separately, it is usually possible to obtain better control results than with a single loop control.

The secondary process variable PV2 is controlled in a secondary control loop. This means that disturbances from this part of the system are compensated before they can affect the quality of the primary process variable PV1. Due to the structure, inner disturbance variables are compensated more quickly since they do not occur in the entire control loop. The setting of the primary controller can then be made more sensitive allowing faster and more precise control with the fixed setpoint SP.

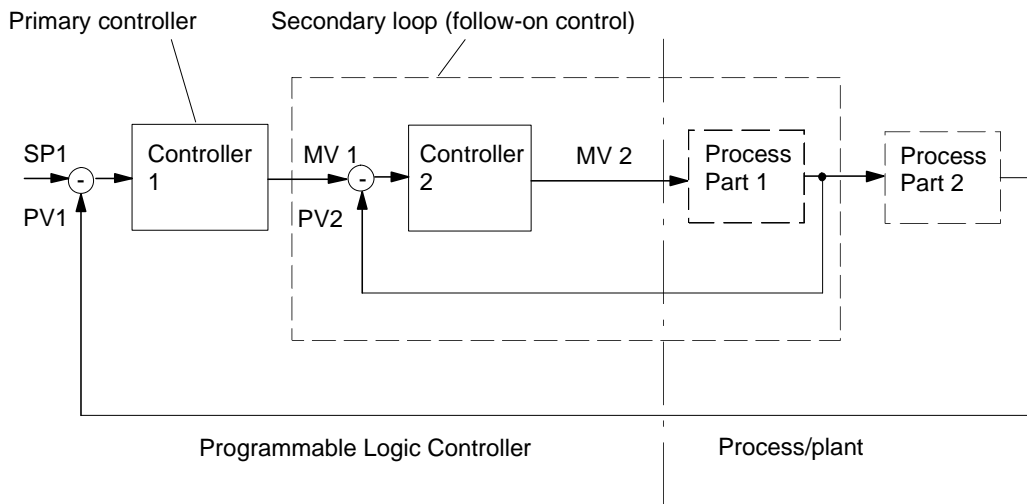


Figure 11-7 Two-Loop Cascade Control System

The controller structure for cascade control contains a controller with a continuous output for controlling the reference input (setpoint) of the secondary loop and a step controller to control the secondary process variable PV2 (secondary controller).

1.4 Structure and Mode of Operation of the PID Control

The controllers that can be implemented with the Standard PID Control are always digital sampling controllers (DDC=direct digital control). Sampling controllers are time-controlled, in other words they are always processed at equidistant intervals (the sampling time or CYCLE). The sampling time or frequency at which the controller is processed can be selected.

The following figure illustrates a simple control loop with the standard controller. This diagram shows you the names of the most important variables and the abbreviations of the parameters as used in this manual.

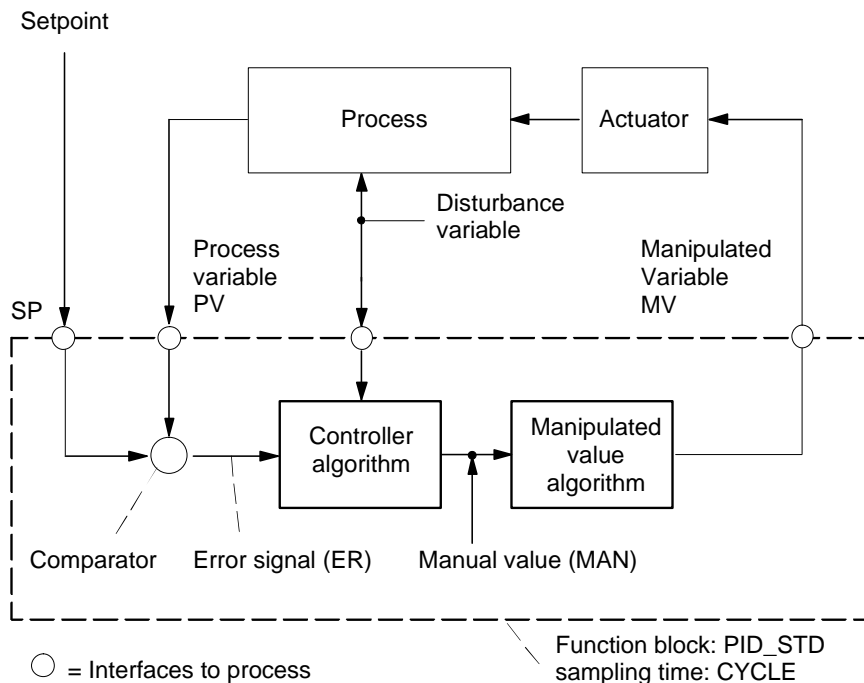


Figure 11-8 Sampling Controller of the Standard PID Control in the Closed Loop

The control functions implemented in the function block PID_STD is pure software controllers. The input and output values of the controllers are processed using digital algorithms on a CPU.

Since the processing of the controller blocks in the processor of the CPU is serial, input values can only be acquired at discrete times and the output values can only be output at defined times. This is the main characteristic of **sampling control**.

1.4.1 Control Algorithm and Conventional Control

The control algorithm on the processor simulates the controller under real-time conditions. Between the sampling instants, the controller does not react to changes in the process variable PV and the manipulated variable MV remains unchanged.

Assuming, however, that the sampling intervals are short enough so that the series of sampling values realistically approximates the continuous changes in the measured variable, a digital controller can be considered as quasi continuous. With the Standard PID Control, the usual methods for determining the structure and setting characteristic values can be used just as with continuous controllers.

This requirement for creating and scaling controllers with the Standard PID Control package is met when the sampling time (CYCLE) is less than 20% of the time constant of the entire loop.

If this condition is met, the functions of the Standard PID Control can be described in the same way as those of conventional controllers. The same range of functions and the same possibilities for monitoring control loop variables and for tuning the controller are available.

1.4.2 The Functions of the "Standard PID Control"

The following diagrams illustrate the preconfigured controller structures of the Standard PID Control as block diagrams. The following figure represents the continuous controller with the signal processing branches for the process variable and setpoint, the controller and the manipulated variable branch. You can see which functions must be implemented after the signal conditioning at the input and which are not required.

The range of functions of the "Standard PID Control" is rigid, but can be extended by a user-defined function (FC) in each of the signal processing branches.

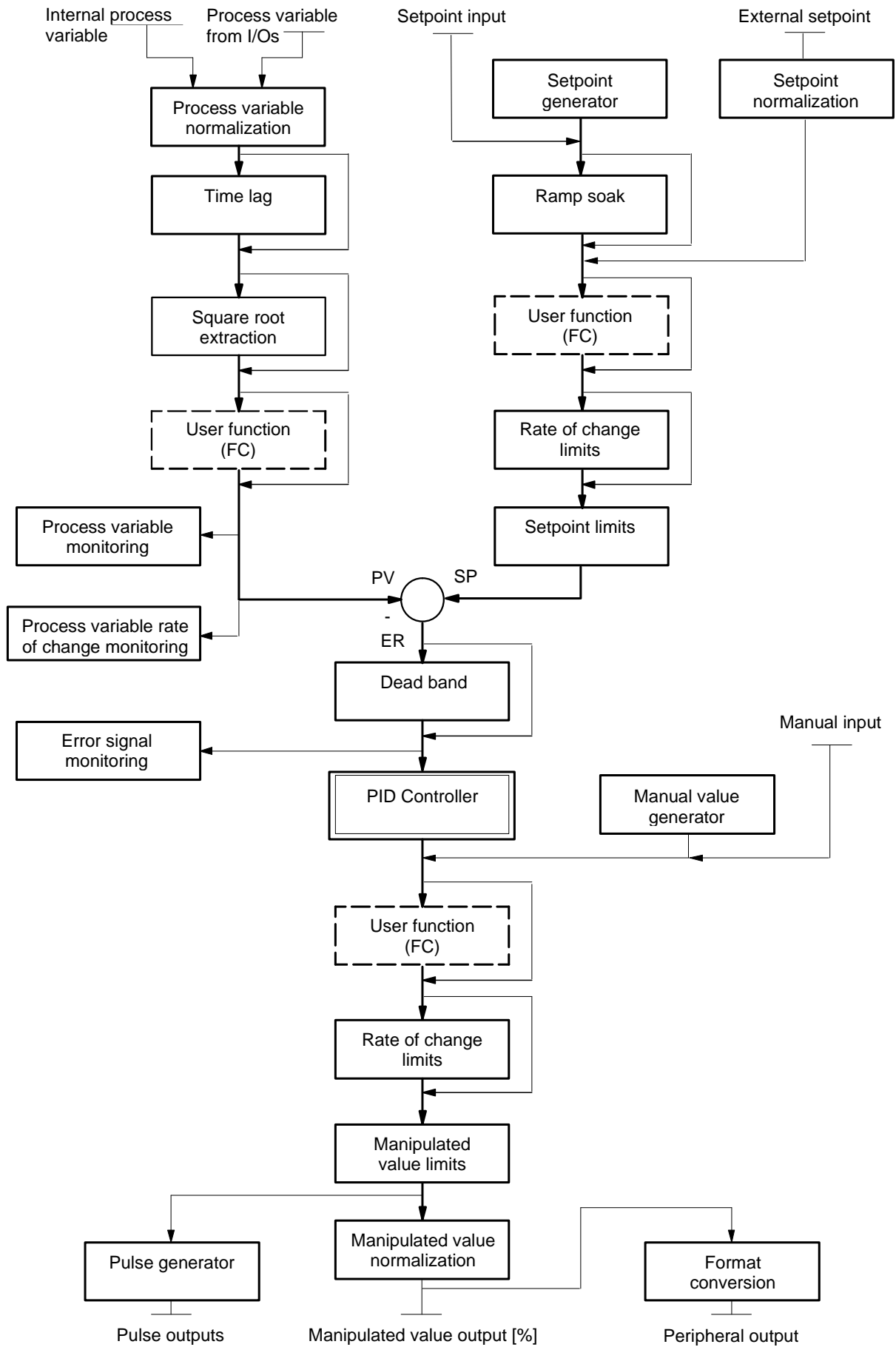


Figure 11-9 equence of Functions of the Standard PID Control (continuous controller)

1.5 Signal Processing in the Setpoint Branch

- **Fixed setting of the setpoint value (SP_GEN)**
With fixed setpoint controllers, the setpoint is selected using a switch at the setpoint generator SP_GEN and is then fixed.
- **Setpoint setting according to a time-controlled program (RMP_GEN)**
When controlling processes with different setpoints set according to a time-controlled program, the ramp soak function generates the required curve for the reference input and influences the process so that the process variable changes according to a defined profile.
- **Change limitation for the reference input(ROC_GEN)**
The conversion of setpoint step changes to a ramp-shaped increase or decrease in the reference input prevents large input changes to the process. The ROC_GEN function limits the setpoint rate of change separately for the up rate and down rate and for positive and negative values in the reference input.
- **Absolute value limitation for the reference input (LIMIT)**
To prevent illegal process states occurring, the setpoint is limited by high and low limits (LIMIT).
- **Delay of the process variable (LAG1_GEN)**
To reduce the effects of noise on process signals, a first order time lag is used in the process variable branch. This function dampens the analog process variable more or less depending on the time constant TMLAG. Disturbance signals are therefore effectively suppressed. Overall, however, the time constant of the total control loop is increased, in other words, the control action becomes slower.
- **Extracting the root of the process variable (SQRT_NORM)**
When the relationship of the measured signal to the physical value is quadratic (flow measurement using a differential flow meter) the process variable must be linearized by extracting the root (square root algorithm). Only a linear value can be compared to the linear setpoint for the flow and processed in the control algorithm. For this reason, the SQRT_NORM function element can be included in the process value branch as an option.
- **Monitoring the Process Variable Rate of Change (CHG_ALM)**
If the rate of change of the process variable is extremely high or too high, this points to a dangerous process state to which the programmable logic controller may have to react. For this reason, the CHG_ALM function generates alarm signals if selectable rates of change (positive or negative) are detected in the process variable. The alarm signals can then be further processed to suit the particular situation.
- **Monitoring the Absolute Value of the Process Variable and Error Signal (LIM_ALM)**
The limit values are set for the process variable and the error signal are monitored by the LIM_ALM function.
- **Superimposing by Signal Noise (DEADBAND)**
To filter out noise on the channels of the process variable or the external reference input, the error signal passes through a selectable dead band component. Depending on the amplitude of the noise, the dead band width can be selected for the signal transmission. Falsification of the transmitted signal must, however be accepted as a side effect of the selected dead band.

1.6 Signal Processing in the PID Controller

- **Fixed Setting of the Manual Value (MAN_GEN)**
In the manual mode (open loop), the manipulated value is selected at the manual value generator MAN_GEN using a switch and is fixed.
- **Change Limitation of the Manipulated Variable (ROC_GEN)**
Converting extremely fast step changes in the manipulated variable into a ramp-shaped rise or fall in the manipulated variable prevents sudden changes in the input to the process. The function (ROC_GEN) limits the manipulated value rate of change both up and down.
- **Absolute value limitation for the Manipulated Variable (LIMIT)**
To avoid illegal process states or to restrict the movement of an actuator, the upper and lower limits of the range of the manipulated variable are set with LIMIT.
- **Forming the Binary Actuating Signal (THREE_STP_GEN)**
Depending on the sign of the error signal, the three-step switch THREE_STP_GEN generates a positive or negative output pulse via the pulse shaping stage.

2. Configuring and Starting the Standard PID Control

2.1 Defining the Control Task

Before you implement a control loop using the Standard PID Control package, you must first clarify the technical aspects of the process you want to automate, the programmable logic controller you will be using and the operating and monitoring environment. To specify the task in detail, you therefore require the following information:

- 1- You need to know the process you want to control, in other words the characteristic data of the process (gain, equivalent time constant, disturbance variables etc.).
- 2- You must choose the CPU on which you want to install the Standard PID Control.
- 3- You must define the signal processing and monitoring functions along with the basic functions of the controller.

Since the Standard PID Control package creates software controllers based on the standard function blocks (for example PID_STD) from the range of Intelart Studio control blocks, you should be familiar with handling those blocks and with the structure of I4PLCs user programs.

Although the functions of the implemented controller are defined solely by assigning parameters, the connection of the controller block to the process I/Os and its integration in the call system of the CPU requires knowledge that cannot be dealt with within the scope of this manual.

You require the following information:

- 1- Working with I4PLCs
- 2- The basics of programming with Intelart Studio
- 3- Data about the programmable logic controller you are using

There are almost no restrictions in terms of the type and complexity of the processes that can be controlled with the Standard PID Control. Providing the system is a single input-single output system without a derivative transfer action and without all-pass components, all process types whether self-regulating processes or not, in other words without or with I components can be controlled (following figure).

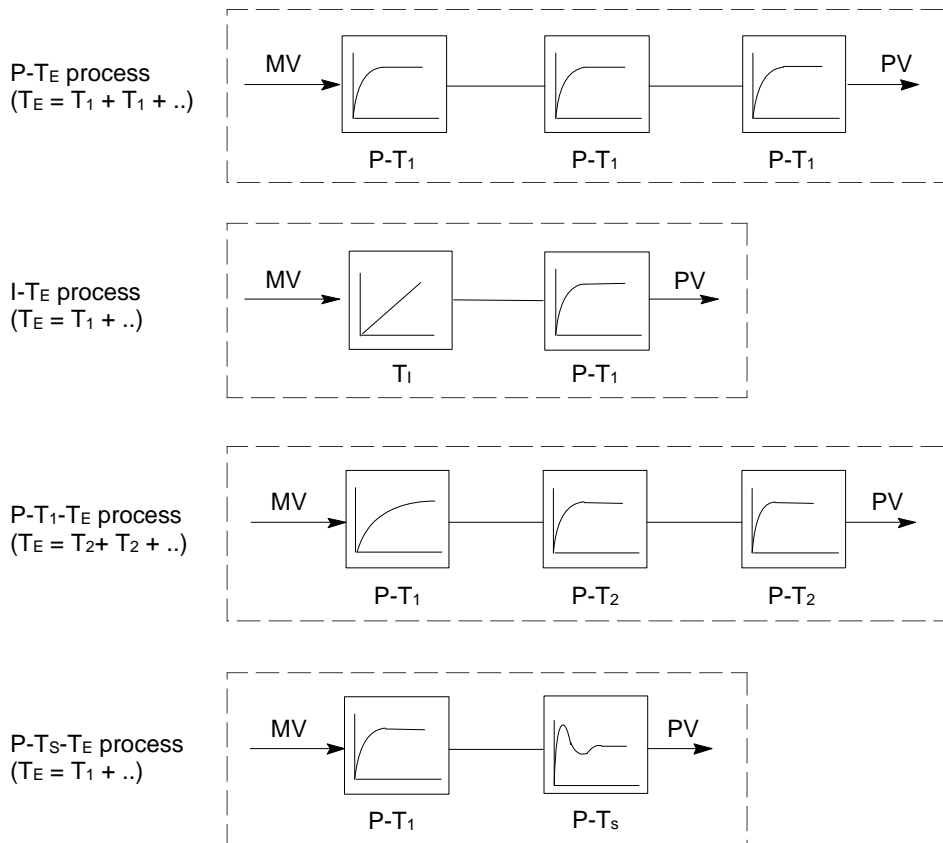


Figure 11-10 Types of Process that can be Controlled with Standard PID Control

The process variable (PV) to be processed by the Standard PID Control is always an analog physical variable (voltage, current, resistance etc.) that is digitized by an expansion analog input module and converted to the uniform Real I/O signal.

The values of these signals are saved in memory cells or areas of the CPU user memory. These areas can be addressed using absolute addresses or using symbolic addresses after making the appropriate entries in the symbol table of the CPU.

2.2 Type of Actuator

To select a suitable configuration for the Standard PID Control, the type of actuator used to influence the process variable is important. The type of signal required by the actuator determines the way in which signals are output in the manipulated variable branch (continuous or discontinuous).

In the great majority of cases, some form of valve will be used to adjust material or energy flow. Different actuating signals are required depending on the drives used to adjust these valves.

1- Proportional actuators with a continuous actuating signal.

The opening of an orifice, the angle of rotation or a position is adopted proportional to the value of the manipulated variable, in other words within the actuating range, the manipulated variable operates in an analog manner on the process. The actuators in this group include pneumatic diaphragm actuators and electro-mechanical actuators with position feedback signals with which a positioning control loop can be created.

2- Proportional actuators with a pulse-width modulated signal.

With these actuators, a pulse signal is output with a length proportional to the value of the manipulated variable at the sampling time intervals. This means that the actuator (for example a heating resistor or heat exchanger) is switched on for a length of time depending on the manipulated variable. The actuating signal can be either monopolar representing the states on or off or bipolar, representing for example the values open/close, forwards/backwards, accelerate/decelerate.

3- Actuators with an integral action and three-step actuating signal.

Actuators are often driven by motors in which the duration of the "on" time is proportional to the travel of the valve plug. Despite different designs, these actuators all share the same characteristic in that they correspond to an integral action at the input to the process. The Standard PID Control with a step output

provides the most economical solution to designing control loops including actuators with an integral action.

💡 TIP

The manipulated variables are represented as digital numerical values in the floating point or peripheral (I/O) format or as binary signal states. Depending on the actuator being used, expansion modules must always be connected to the output to convert the signals to the required type and to provide the required actuating energy.

2.3 Generating the Control Project Configuration

Now that you have worked through the required control and monitoring, this section now shows you the step-by-step implementation of these functions. We recommend that you create your configuration following the steps outlined below (checklist):

- 1- Select the controller blocks or block configuration required for your controller structure. Select and copy a configuration example closest to the configuration you want to implement.
- 2- Configure the required controller by including or omitting preconfigured functions or by including your own.
- 3- Select the sampling time and calls of the control loop:
 - Specify the startup response with Startup OB
 - Decide on the sampling time and priority class, if necessary, change the call interval of the periodic interrupt OB
 - Configure the loop scheduler to suit the number of loops on the CPU
- 4- Assign parameters and use the conversion functions for the measuring range and zero point adaptation of the input/output signals:
 - Normalization of the external setpoint
 - Normalization of the external process variable
 - Manipulated value denormalization
- 5- Configure the setpoint branch:
 - Setpoint generator
 - Ramp soak
 - Limits of the setpoint rate of change
 - Limits of the absolute values of the setpoint
- 6- Configure the process variable branch:
 - Process variable time lag
 - Square root extraction
 - Monitor the absolute values of the process variable
 - Monitoring the rate of change of the process variable
- 7- Configure error signal generation:
 - Dead band of the error signal
 - Monitoring the error signal for absolute values
- 8- Configure the manipulated value branch for continuous controllers:
 - Manual value generator
 - Limits of the rate of change of the manipulated value
 - Limits of the absolute values of the manipulated value
- 9- Configure controller:
 - PID controller structure and PID parameters

- Operating point for P and PD controllers
 - Feedforward control
- 10- If necessary, include extra functions in the form of a user FC in the setpoint, process variable and/or manipulated value branch.
- 11- Interconnect the block inputs and outputs of the configured standard controller with the process I/Os:
- Program the connections of the inputs/outputs with the absolute or symbolic I/O addresses in the user memory of the CPU.

2.4 The Sampling Time CYCLE

2.4.1 The Sampling Time: CYCLE

The sampling time is the basic characteristic for the dynamic response of the Standard PID Control. This decides whether or not the controller reacts quickly enough to process changes and whether the controller can maintain control in all circumstances. The sampling time also determines the limits for the time-related parameters of the Standard PID Control.

Selecting the sampling time is a compromise between several, often contradictory requirements. Here, it is only possible to specify a general guideline.

- The time required for the CPU to process the control program, in other words to run the function block, represents the lowest limit of the sampling time.
- The tolerable upper limit for the sampling time is generally specified by the process dynamics. The process dynamics is, in turn, characterized by the type and the characteristics of the process.

2.4.2 Equivalent System Time Constant

The most important influence on the dynamics of the control loop is the equivalent system time constant (T_E) that can be determined after entering a step change MV by recording the unit step response at the system input. The system value T_E represents a useful approximation of the effective time lag caused by several P-T₁, P-T_S and T_i elements in the loop. If, for example the same PT₁-elements are connected in series, it is the sum of the single time constants.

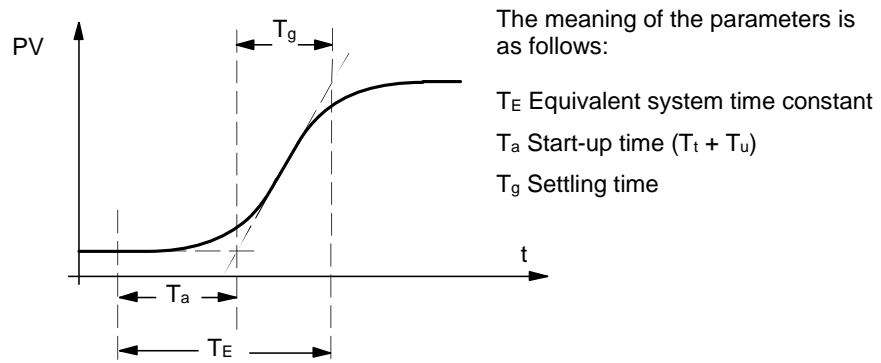


Figure 11-11 Determining the Equivalent System Time Constant T_E

2.4.3 Sampling Time Estimate

If a minimum speed is required for the control, you can specify a maximum sampling time $CYCLE_{max}$.

With P- T_E processes in which the first delay element is predominant and $T_1 \geq 0.5 T_E$ make sure that:

$$CYCLE_{max} \leq 0.1 * T_E$$

For all other P- T_E -processes:

$$CYCLE_{max} \leq 0.2 * T_E \text{ is adequate}$$

2.4.4 Rule of Thumb for Selecting the Sampling time

Experience has shown that a sampling time of approximately 1/10 of the time constant T_{EG} determining the step response of the closed loop produces results comparable with the conventional analog controller. The total time constant of the closed loops is obtained in a way similar to that shown in Figure 11-11, by entering a setpoint step change and evaluating the settling of the process variable.

$$CYCLE = \frac{1}{10} T_{EG}$$

2.5 How the Standard PID Control is Called

Depending on the sampling time of the specific controller, the controller block must be called more or less often but always at the same time intervals. The operating system of the I4PLC calls the periodic interrupt organization block at the specified intervals.

If you require several controllers or controllers with large sampling times, you should use the loop scheduler mechanism (LP_SCHED).

2.6 Range of Values and Signal Adaptation (Normalization)

2.6.1 Internal Numerical Representation

When the algorithms in the function blocks of the Standard PID Control are processed, the processor works with numbers in the floating point format (REAL).

The floating point numbers have the single format complying with ANSI/IEEE standard 754-1985:

Format: DD (32 bits)
 Range of values: $-3.37 * 10^{38} \dots -8.43 * 10^{-37}$ and $8.43 * 10^{-37} \dots 3.37 * 10^{38}$

This range is the total range of values for parameters in the REAL format. To avoid limits being exceeded during processing, the input signal SP which is an analog physical value is defined as a technical range of values:

Techn. Range of values: $-10^5 \dots +10^5$

2.6.2 Signal Adaptation

The normalization function at the input for the external setpoint allows any characteristic curve of transmitters or sensors to be adapted to the physical range of values of the Standard PID Control.

3. Signal Processing in the Setpoint/Process Variable Channels and PID Controller Functions

3.1 Average Value Generator (AVG_GEN)

Table 11-1 AVG_GEN instruction

LAD/ FBD	Description
	Moving average by an external buffer and selectable averaging samples count.

Supported Properties: None

Table 11-2 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode. <ul style="list-style-type: none"> False = Fills the buffer with INV value and outs the INV on OUT True = Running moving average
INV	Real	Input variable
MN_N	Int	Number of mean (average) elements
BUF	Real[*]	Ring buffer
OUT	Real	Average output

3.2 Rate of Change Alarm Generator (CHG_ALM)

Table 11-3 CHG_ALM instruction

LAD/ FBD	Description
	<p>The CHG_ALM function monitors limits for the rate of change of any process variable.</p> <p>The numerical values for the rate of change limits are set at the input parameters for “up rate” and “down rate” in the positive and negative ranges of the process variable. The rate of change is an up or down rate as a percentage per second.</p> <p>If the rate of change of the process variable exceeds these limits, the output signal bits QURLMP to QDRLMN are set.</p>

Supported Properties: None

Application

If the rate of change in a process variable is too fast (for example motor speed, pressure, level, temperature etc.), illegal or dangerous situations can occur in the process or plant. Here, the CHG_ALM function is used to make sure that the process variable does not exceed or fall below a permitted range of change or slope. Limit violations are detected and signaled to allow a suitable reaction.

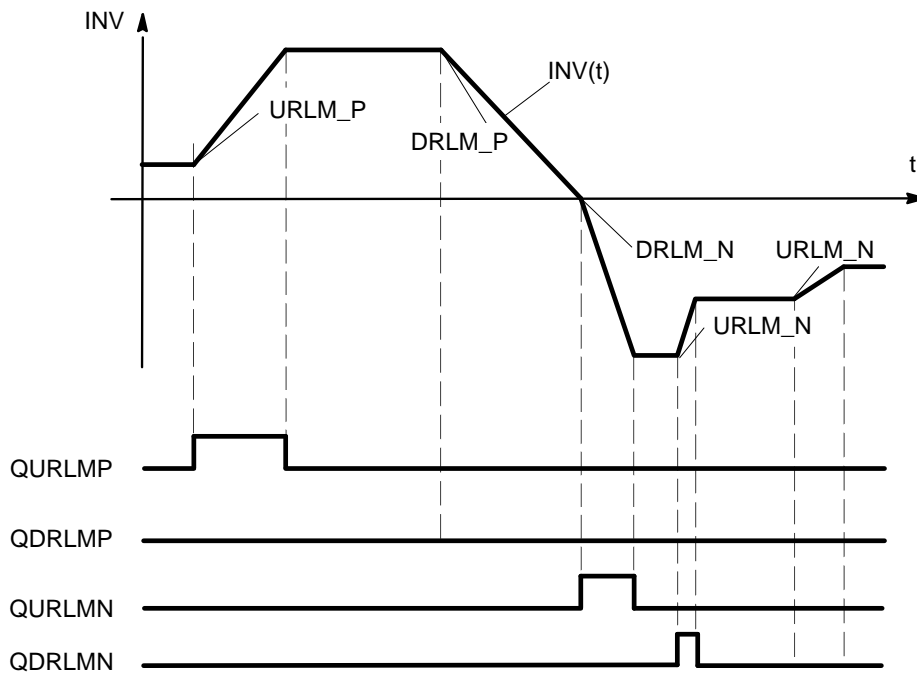


Figure 11-12 Monitoring the Rate of Change (Slope) of a Process Variable INV(t)

Table 11-4 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
INV	Real	Input variable
URLM_P	Real	INV rise limit in the positive range. $INV > 0$ and $ INV $ rising
DRLM_P	Real	INV fall limit in the positive range. $INV > 0$ and $ INV $ falling
URLM_N	Real	INV rise limit in the neg. range. $INV < 0$ and $ INV $ rising
DRLM_N	Real	INV fall limit in the neg. range. $INV < 0$ and $ INV $ falling
MN_N	Int	Number of mean (average) elements. $0 < MN_N \leq 8$
OUT	Real	Current change value
QURLMP	Bool	Rise limit in the positive range alarm
QDRLMP	Bool	Fall limit in the positive range alarm
QURLMN	Bool	Rise limit in the negative range alarm
QDRLMN	Bool	Fall limit in the negative range alarm

3.3 Cycle Time Calculator (CYC_TM)

Table 11-5 CYC_TM instruction

LAD/ FBD	Description
	Calculates the elapsed time of its last execution.

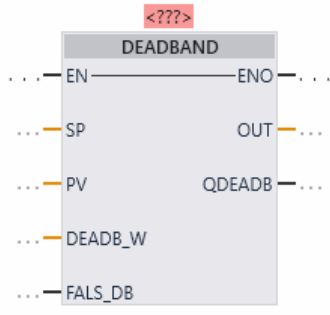
Supported Properties: None

Table 11-6 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
OUT	Time	Calculated cycle time

3.4 Filtering Signal Function (DEADBAND)

Table 11-7 DEADBAND instruction

LAD/ FBD	Description
	<p>The DEADBAND function is a selectable band in which small fluctuations in the input variable around a specified zero point are suppressed. Outside this band, the error signal OUT rises or falls in proportion to the input value. You can specify the width of the DEADBAND using the parameter DEADB_W. The DEADBAND width can only have positive values.</p> <p>If the input variable is within the DEADBAND, the value 0 is output (error signal = 0). The output only rises or falls by the same values as the input variable inv only when the input variable leaves this DEADBAND. This also falsifies the transferred signal when it is outside the DEADBAND. This is, however, an acceptable compromise to avoid step changes at the limits of the DEADBAND. The amount to which the signal is falsified corresponds to the value DEADB_W and can therefore be checked easily.</p>

Supported Properties: None

Application

If the process variable or the setpoint is affected by higher frequency noise and the controller is optimally set, the noise will also affect the controller output. This can, for example, lead to large fluctuations in the manipulated value at high control again when the D action is activated. This function reduces noise in the error signal of the controller in the settled state and thus reduces unwanted oscillation of the controller output.

The DEADBAND operates according to the following functions:

Modified PV = SP - PV + DEADB_W where $SP - PV < -DEADB_W$

Modified PV = 0 where $-DEADB_W \leq SP - PV \leq +DEADB_W$

Modified PV = SP - PV + DEADB_W where $SP - PV > +DEADB_W$

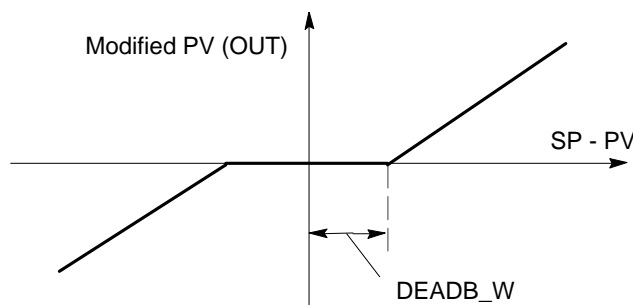


Figure 11-13 Filtering Noise Affecting the Error Signal (SP – PV) using a DEADBAND

Table 11-8 Data types for the parameters

Parameter	Data type	Description
SP	Real	Setpoint
PV	Real	Process variable
DEADB_W	Real	Dead band width. DEADB_W >=0

FALS_DB	Bool	Falsify error outside dead band. False = Disabled, True = Enabled
OUT	Real	Modified PV
QDEADB	Bool	Control error is within dead band

3.5 Unsigned Int to Signed Int Encoder (ENCODER)

Table 11-9 ENCODER instruction

LAD/ FBD	Description
	Converts a 16-bit unsigned counter value to 32-bit unsigned value.

Supported Properties: None

Table 11-10 Data types for the parameters

Parameter	Data type	Description
INV	UDInt	Input value from 16-bit encoder
OUT	DInt	32-bit encoder value

3.6 First In First Out (FIFO)

Table 11-11 FIFO instruction

LAD/ FBD	Description
	The full form of FIFO is First In, First Out. FIFO is a method of organizing, handling, and manipulating the data structure of elements in a computing system. It's a type of data handling which prioritizes the processes that come first- meaning, it will first remove or append those elements that came first.

Supported Properties: None

Table 11-12 Data types for the parameters

Parameter	Data type	Description
N	Int	Maximum depth after reset
QUEUE	Bool	Basic queue operations
PEEK	Bool	Basic queue operations
R1	Bool	Over-riding reset
IN	Variant	Input to be queued

OUT	Variant	First element data
BUFFER	Variant[*]	External array
EMPTY	Bool	Stack empty
OFLO	Bool	Stack overflow

3.7 Asymmetric Hysteresis Generator (HYST_GEN)

Table 11-13 HYST_GEN instruction

LAD/ FBD	Description
	<p>The Asym hysteresis function block provides an asymmetric hysteresis boolean output driven by the difference of two floating point (REAL) inputs XIN1 and XIN2.</p>

Supported Properties: None

Table 11-14 Data types for the parameters

Parameter	Data type	Description
XIN1	Real	Input 1
XIN2	Real	Input 2
EPS_H	Real	High epsilon
EPS_L	Real	Low epsilon
Q	Bool	$XIN1 > XIN2 + EPS_H = 1$, $XIN1 - EPS_L < XIN2 = 1$

3.8 Damping the Process Variable (LAG1_GEN)

Table 11-15 LAG1_GEN instruction

LAD/ FBD	Description
	<p>By incorporating a time delay, higher frequency fluctuations in the process variable signal can be damped so that they are excluded from the processing in the control algorithm in particular to avoid affecting the derivative action. The amount of signal damping is determined by the time constant TMLAG. The damping effect is achieved by a first order time lag algorithm.</p>

Supported Properties: None

Application

The LAG1_GEN function is used as a delay element for the process variable. This can be used to suppress disturbances.

By incorporating a time delay, higher frequency fluctuations in the process variable signal can be damped so that they are excluded from the processing in the control algorithm in particular to avoid affecting the derivative action. The amount of signal damping is determined by the time constant TMLAG.

The damping effect is achieved by a first order time lag algorithm.

The transfer function in the Laplace transform is as follows:

$$\frac{\text{outv}(s)}{\text{MP4}(s)} = \frac{1}{(1 + \text{TMLAG} * s)} \quad \text{where } s = \text{Laplace variable}$$

The step response in the time domain is as follows:

$$\text{outv}(t) = \text{MP4}(0) (1 - e^{-t/\text{PV_TMLAG}})$$

Legend:

MP4(0) the size of the process variable jump at the input

outv(t) the output value

TMLAG the delay time constant

t time

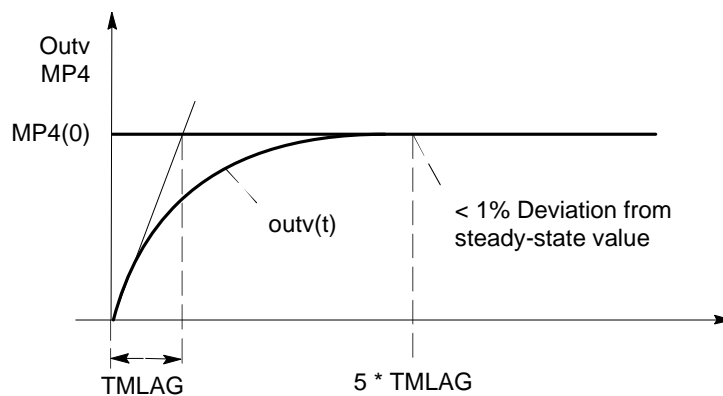


Figure 11-14 Time lag smoothing diagram

Conditions for Parameter Assignment

If $\text{TMLAG} \leq 0.5 * \text{CYCLE}$, there is no lag in effect.

A sampling time (CYCLE) of less than a fifth of the time lag is necessary to achieve a time lag approaching the analog response.

Table 11-16 Data types for the parameters

Parameter	Data type	Description
INV	Real	Input variable
TMLAG	Time	Input variable time lag
DF_OUT	Real	Default output variable
DF_OUT_ON	Bool	Output default value on
RST_ON	Bool	Restart
CYCLE	Time	Sample time
OUT	Real	Output variable

3.9 Monitoring a Process Variable Limits (LIM_ALM)

Table 11-17 LIM_ALM instruction

LAD/ FBD	Description
	<p>The LIM_ALM function monitors four selectable limits in two tolerance bands for a process variable INV(t). If the limits are reached or exceeded, the function signals a warning at the first limit and an alarm at the second limit.</p> <p>The numerical values of the limits are set in the input parameters for "Warning" and "Alarm". If the process variable (INV) exceeds or falls below these limits, the corresponding output bits QH_ALM, QH_WRN, QL_WRN and QL_ALM are set.</p> <p>To prevent the signal bits "flickering" due to slight changes in the input value or due to rounding errors, a hysteresis HYS is set. The hysteresis must pass the process variable before the messages are reset.</p>

Supported Properties: None

Application

Illegal or dangerous states can occur in a system if process values (for example motor speed, pressure, level, temperature etc.) exceed or fall below critical values. In such situations, the LIM_ALM function is used to monitor the permitted operating range. Limit violations are detected and signaled to allow a suitable reaction.

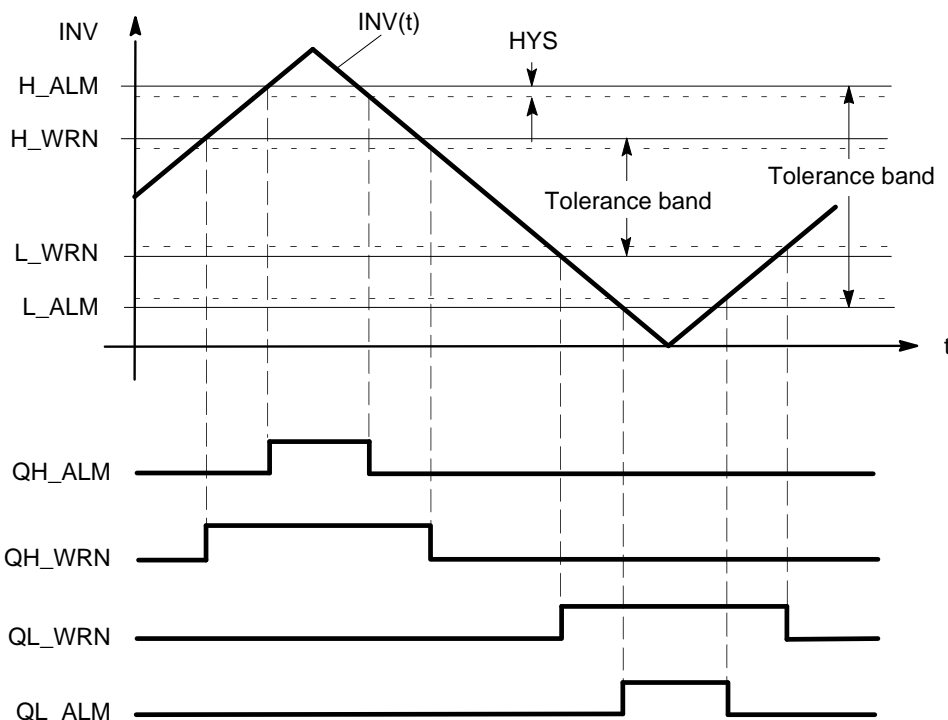


Figure 11-15 A Process Variable INV – Monitoring the Limit Values

Table 11-18 Data types for the parameters

Parameter	Data type	Description
INV	Real	Input variable
H_ALM	Real	Upper INV limit 'alarm'
H_WRN	Real	Upper INV limit 'warning'

L_ALM	Real	Lower INV limit 'alarm'
L_WRN	Real	Lower INV limit 'warning'
HYS	Real	INV hysteresis
QH_ALM	Bool	High limit alarm
QH_WRN	Bool	High limit warning
QL_WRN	Bool	Low limit warning
QL_ALM	Bool	Low limit alarm

3.10 Loop Scheduler (LP_SCHED)

Table 11-19 LP_SCHED instruction

LAD/ FBD	Description
	<p>The "LP_SCHED" function reads the parameters specified by you calculates the variables required to schedule the loops.</p> <p>You should call the "LP_SCHED" FC in a periodic interrupt OB. Afterwards you must program a conditional call for all the corresponding control loops in the same OB. The condition for calling the individual control loops is determined by the "SCHED" Bool array. During operation you can disable the call of individual control loops manually and furthermore reset individual control loops.</p>

Supported Properties: None

Table 11-20 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Input value from 16-bit encoder
TM_B	Time	Time base
SCHED	Bool[*]	Schedule bool array
OUT	Int	Current schedule index

Application

The loop scheduler LP_SCHED is used when the number of periodic-interrupts of a CPU is not enough to realize the desired (various) sampling times. It allows any size of control loops to be called with sampling times which amount to time base (TM_B) of the OB cycles.

Call of the "LP_SCHED" FB in your Program

The "LP_SCHED" FB must be called before all control loop FBs.

Observe the following points when assigning values to the input parameter.

- RUN: When True, the scheduler will be run. When False, the scheduler is disabled and all SCHED bits are in reset mode
- TM_B: At this point enter the interval time of the schedule to be executed.
- SCHED: A Bool array in size of your control loops which you must assign to the FB in order to update its status by the scheduler
- OUT: Current schedule index in run mode. -1 when scheduler is disabled

When you call the control loop FBs you have to interconnect their input parameters EN and CYCLE with the variables SCHED[x] and TM_B of the FB. SCHED [x] contains the trigger Boolean value only for one cycle and is written by the "LP_SCHED" FB at every run.

The following section gives an example for calling the "LP_SCHED" FB and for the conditional call of three functions.

TIP

We recommend you to run LP_SCHED in a periodic interrupt OB instead of a cyclic program OB. When you run the LP_SCHED in a periodic interrupt you must set the interval of the OB to be a chunk of the TM_B

parameter of the LP_SCHED FB. For example, if you want to execute 4 functions in every 200ms intervals in a periodic interrupt, then you must set the Interval property of the OB to a rounded number such as $200\text{ms}/4 = 50\text{ms}$ or another less coefficient number. In cyclic program OB there is no periodic interval and no guarantee to run schedules on a precise time base.

NOTICE

If you set a wrong TM_B value for the LP_SCHED FB, an arranged executions order may be occur.

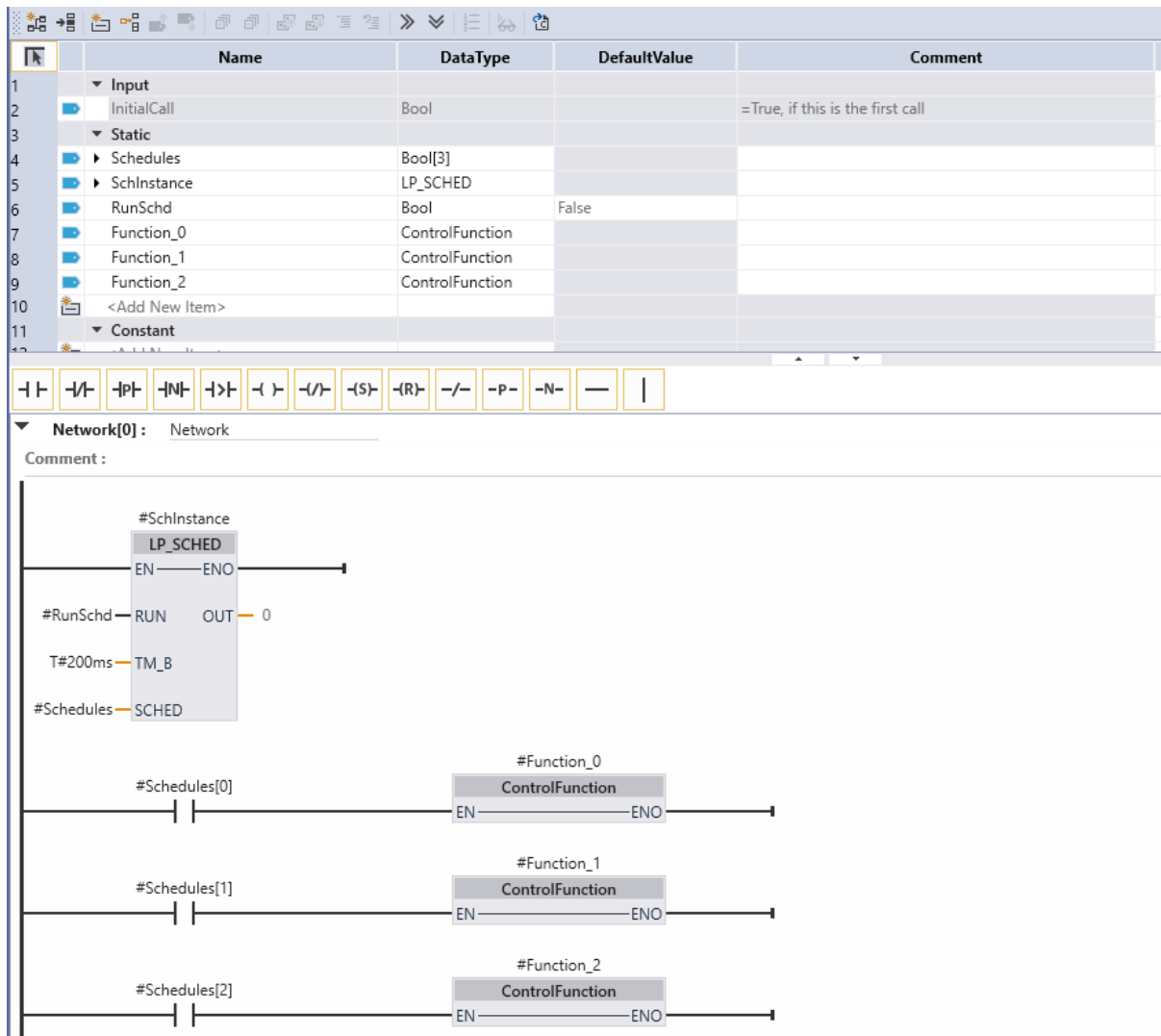


Figure 11-16 Example of LP_SCHED for 3 scheduled functions

3.11 Manual Value Generator (MAN_GEN)

Table 11-21 ENCODER instruction

LAD/ FBD	Description
	<p>The MAN_GEN function generates a value that can be set or modified using switches. The output variable OUT can be increased or decreased step-by-step via the binary inputs OUT_UP and OUT_DN.</p> <p>The range of the setpoint is restricted by the high/low limits H_LM/ L_LM in the value branch. The numerical values of the limits (as percentages) are set using the corresponding input parameters. The signal outputs QH_LM and QL_LM indicate when these limits are exceeded.</p> <p>The rate of change of the output variable depends on the length of time the switches OUT_UP or OUT_DN are activated and on the selected limits as shown below:</p> <p>During the first 3 seconds after setting OUT_UP or OUT_DN:</p> $\frac{d \text{ outv}}{dt} = \frac{H_LM - L_LM}{100 \text{ s}}$ <p>afterwards:</p> $\frac{d \text{ outv}}{dt} = \frac{H_LM - L_LM}{10 \text{ s}}$

Supported Properties: None

Table 11-22 Data types for the parameters

Parameter	Data type	Description
DF_OUT	Real	Default output variable
H_LM	Real	Input variable high limit
L_LM	Real	Input variable low limit
OUT_UP	Bool	Output variable up
OUT_DN	Bool	Output variable down
DF_OUT_ON	Bool	Output default value on
RST_ON	Bool	Restart
CYCLE	Time	Sample time
OUT	Real	Output variable
QH_LM	Bool	Output variable high limit
QL_LM	Bool	Output variable low limit

Application

Using a higher/lower switch, you can adjust the internal setpoint.

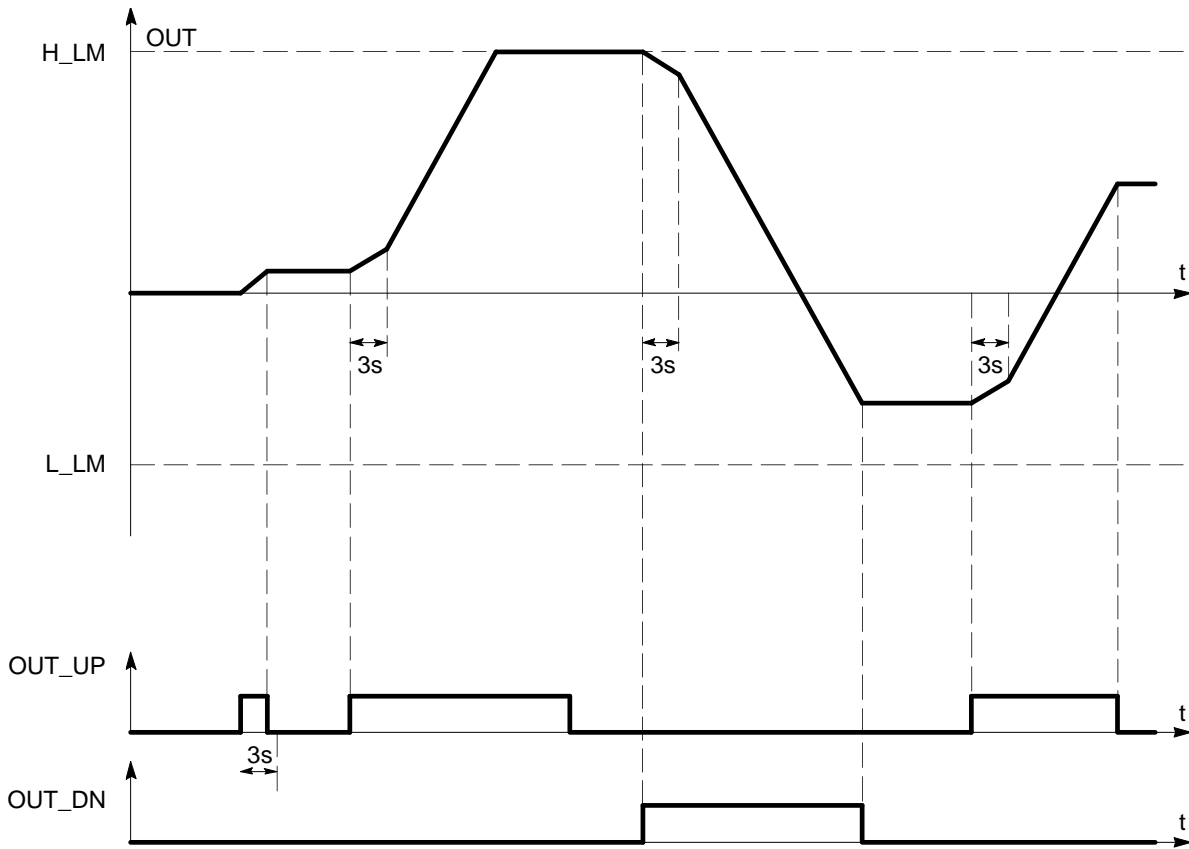


Figure 11-17 Changing OUT as a function of the switches OUT_UP and OUT_DN

3.12 Normalize (NORM)

Table 11-23 NORM instruction

LAD/ FBD	Description
	<p>Normalizes the parameter INV inside the value range specified by the IN_L and IN_H parameters:</p> $OUT = (INV - IN_L) / (IN_H - IN_L), \text{ where } (0.0 \leq OUT \leq 1.0)$

Supported Properties: None

Table 11-24 Data types for the parameters

Parameter	Data type	Description
INV	AnyNum	Input variable
IN_L	AnyNum	Input low limit
IN_H	AnyNum	Input high limit
OUT	AnyNum	Normalized output

3.13 Standard PID (PID_STD)

Table 11-25 PID_STD instruction

LAD/ FBD	Description
	<p>This FB implements a complete PID controller with continuous manipulated variable output with the option of adjusting the manipulated value manually.</p> <p>Subfunctions can be enabled or disabled.</p> <p>Using the FB, you are in a position to control technical processes and systems with continuous input and output variables on I4PLC programmable logic controllers. The controller can be used as a fixed setpoint controller either individually or in multi-loop control systems as a cascade, blending or ratio controller.</p>

Supported Properties: None

Table 11-26 Data types for the parameters

Parameter	Data type	Description
MAN_ON	Bool	Manual value on (variable MAN_MV)
MAN_MV	Real	Manual MV variable
SP	Real	Setpoint
PV	Real	Process variable
KP	Real	Proportional gain
TI	Real	Reset time
TD	Real	Derivative time
I_SEL	Bool	Integral action on
D_SEL	Bool	Derivative action on
CP0	Real	Control parameter 0
CP1	Real	Control parameter 1
CYCLE	Time	Sample time of controller
RST_ON	Bool	Initialize on
MV	Real	Manipulated value
C_DEV	Real	Control deviation (%)
Static Members		
MV_LO	Real	Low limit of mv
MV_HI	Real	High limit of mv
DIST_BND	Real	Disturbance rejection band (%)
STD_BND	Real	Steady state band (%)

DER_N	Real	Derivative gain mode. $8 \leq x \leq 20$: Damper gain for D on PV. $8 > x$: D with time lag
DS_CORR	Real	Dev supp Correction offset (%)
DS_PULUP_DEV	Real	Dev supp Pull up deviation threshold
DS_PULDN_DEV	Real	Dev supp Pull down deviation threshold
DS_PULUP_TRG	Time	Dev supp Pull up time trigger
DS_PULDN_TRG	Time	Dev supp Pull down time trigger
PRP_RT	Real	Proportional rate
INT_RT	Real	Integral rate
DER_RT	Real	Derivative rate
ERROR	Real	Control error
P_SEL	Bool	Proportional action on
C_DIR	Bool	Control direction. 0=Inverse, 1=Direct
RMP_EQ	Bool	Ramp equivalence integration. 0=Disabled, 1=Enabled
SM_INIT	Bool	Smooth initialization. 0=Disabled, 1=Enabled
DS_PULUP	Bool	Deviation suppression Pull up on
DS_PULDN	Bool	Deviation suppression Pull down on
DS_PCT	Bool	Deviation suppression on percentage of error
QINT_ON	Bool	Integrator operation on
QDIST_REJ	Bool	Disturbance rejection activated
QDS_HI_OUT	Bool	Dev supp PV greater than upper band range
QDS_LO_OUT	Bool	Dev supp PV less than lower band range
QPB_OUT	Bool	Error is outside the proportional band

3.13.1 Block Diagram of the Standard Controller

The mode of operation is based on the PID control algorithm of the sampling controller with an analog output signal, if necessary, supplemented by a pulse generator stage for generating pulse-duration modulated output signals for two or three-step controllers with proportional actuators.

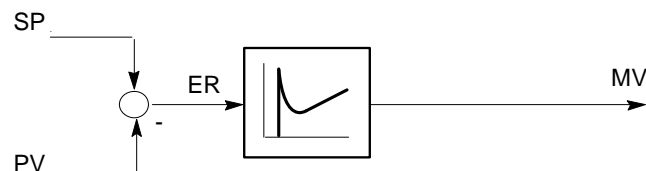


Figure 11-18 Block Diagram of the Controller with Continuous Actuating Signal

3.13.2 Complete Restart/Restart

The PID_STD function block has an initialization routine that is run through when the input parameter RST_ON = True is set.

3.13.3 Integral action (INT)

When the controller starts up, the integrator is set to the initialization value MAN_MV (if SM_INIT= True) and the integral action is output at the MV output. When it is called by a periodic interrupt, it starts at this value.

All other outputs are set to their default values.

3.13.4 Manual Mode and Changing Modes

In addition to the “automatic” mode with the output switched to the output of the PID algorithm (MV), the Standard PID Control also has a manual mode in which the manipulated variable can be influenced manually.

Using the parameter MAN the manipulated variable can be adjusted externally either setting the value manually or by the user program setting the value when MAN_ON = True. The input value MAN is limited to the manipulated variables MV_HI upper) and MV_LO (lower).

3.13.5 Automatic Mode

If MAN_ON = False is selected, the manipulated value of the PID algorithm is connected to the output. In manual mode (MAN_ON = True) the integral components of the controller are disabled so that the controller begins with a sensible manipulated variable when changing over to automatic mode only when SM_INIT = True.

3.13.6 Limiting the Absolute Value of the Manipulated

The operating range, in other words the range through which the actuator can move within the permitted range of values, is determined by the range of the manipulated variable. Since the limits for permitted manipulated values do not normally match the 0% or 100% limit of the manipulated value range, it is often necessary to further restrict the range.

To avoid illegal statuses occurring in the process, the range for the manipulated variable has an upper and lower limit in the manipulated variable branch MV_LO and MV_HI.

3.13.7 Control Algorithm and Controller Structure

Within the cycle of the configured sampling time, the manipulated variable of the continuous controller is calculated from the error signal in the PID algorithm. The controller is designed as a parallel structure. The proportional, integral and derivative actions can be deactivated individually.

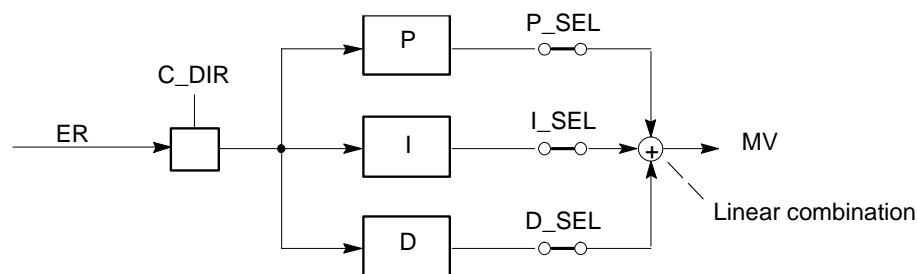


Figure 11-19 Control Algorithm of the Standard PID Control (Parallel Structure)

3.13.8 Defining the Controller Structure

To define an effective controller structure, there are three switches available. The setting of this structure switch is carried out in the configuration tool by selecting the P, I and D actions.

Table 11-27 Selecting the Controller Structure

Mode	Switch	P_SEL	I_SEL	D_SEL
P controller		True	False	False
PI controller		True	True	False
PD controller		True	False	True
PID controller		True	True	True

Reversing the Controller Functions

You can reverse the controller from

- Rising process variable PV(t) → falling manipulated variable MV(t) (Inverse mode)

to the

- Rising process variable PV(t) → rising manipulated variable MV(t) (Direct mode)

by setting a True value for C_DIR switch. C_DIR decides the direction of the control action of the continuous controller.

3.13.9 P Controller

In a P controller, the I and D actions are disabled. (I_SEL and D_SEL = False).

This means that if the error signal ER is 0, the output signal MV is also 0. If an operating point 0 is required, in other words a numerical value for the output signal when the error signal is zero, the I action must be activated.

The step response of the P controller in the time domain is as follows:

$$MV(t) = KP * ER(t)$$

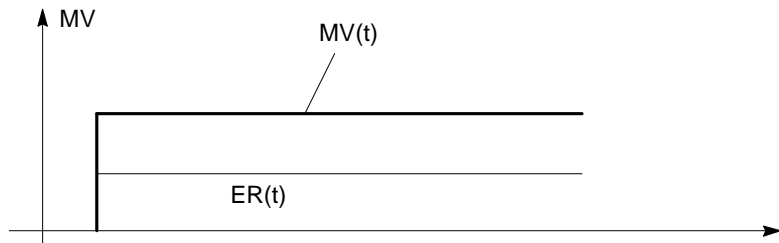


Figure 11-20 Step Response of the P Controller

3.13.10 PI Control

In a PI controller, the D action is disabled. ($D_SEL = False$). A PI controller adjusts the output variable MV using the I action until the error signal ER becomes zero. This only applies when the output variable does not exceed the limits of the manipulated value.

The step response in the time domain is as follows:

$$MV(t) = KP * ER(t) * (1 + \int dt / TI)$$

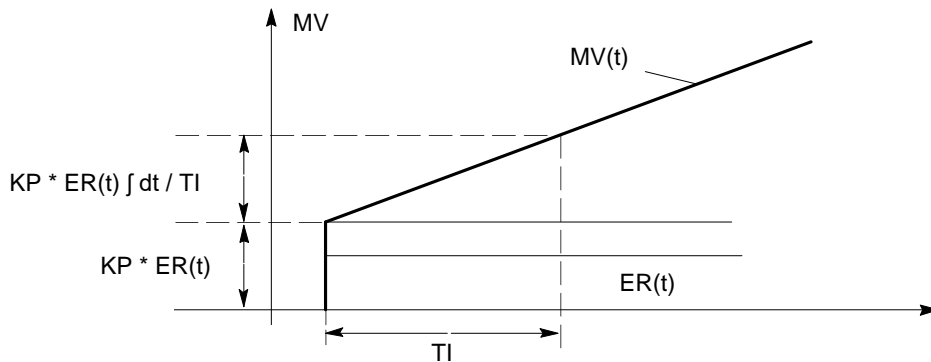


Figure 11-21 Step Response of the PI Controller

To allow a smooth changeover from the manual mode to the automatic mode of the PI controller, the output signal is switched to the internal memory of the integrator when the manipulated variable is being adjusted manually (SM_INIT must be True).

To achieve a purely integrating control action disable the P action with P_SEL .

3.13.11 PD Controller

In the PD controller, the I action is deactivated ($I_SEL = False$). This means that if the error signal ER is zero, the output signal MV is also zero. If an operating point 0 is required, in other words a numerical value must be set for the output signal when the error signal is zero, then the I branch must be activated.

With the I action, an operating point 0 can be specified for the P controller by setting an initialization value. To do this, set switch ' SM_INIT ' and ' I_SEL ' to True.

The PD controller forms the input value $ER(t)$ proportional to the output signal and adds the D action formed by differentiating $ER(t)$ that is calculated with twice the accuracy according to the trapezoidal rule (Padé approximation). The time response is determined by the derivative action time TD. To damp signals and to suppress disturbances, a first order time lag (adjustable time constant: DER_N) is integrated in the algorithm for forming the D action. Generally, a small value ($DER_N < 8$) is adequate for DER_N to achieve a successful outcome. If $DER_N > 2/CYCLE$ is configured, the time lag is disabled.

The step response in the time domain is as follows:

$$MV(t) = KP * ER(t) * (1 + DER_N * TD * EXP(t * DER_N))$$

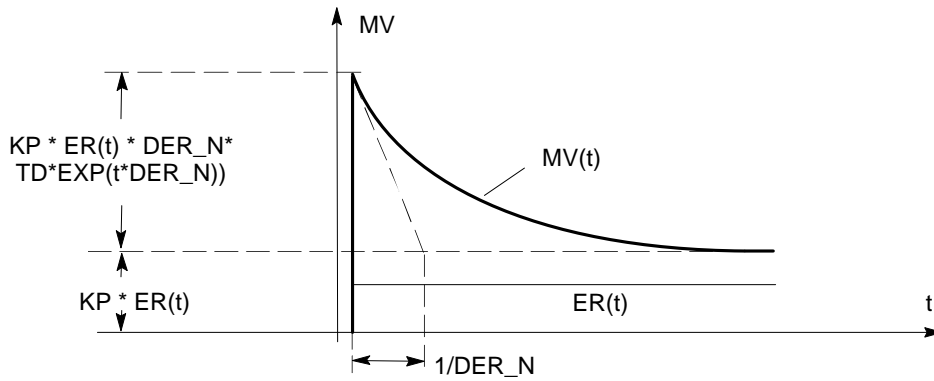


Figure 11-22 Step Response of the PD Controller

3.13.12 PID Controller

In a PID controller, the P, I and D actions are activated ($P_SEL, I_SEL, D_SEL = True$). A PID controller adjusts the output variable MV using the I action until the error signal ER becomes zero. This only applies when the output variable does not exceed the limits of the manipulated value. If the manipulated variable range limits are exceeded, the I action retains the value that was set when the limit was reached (anti reset wind-up). The PID controller forms the input value ER (t) proportional to the output signal and adds the actions formed by differentiating and integrating ER (t) that are calculated with twice the accuracy according to the trapezoidal rule (Padé approximation). The time response is determined by the derivative action time TD and the reset time TI.

To damp signals and to suppress disturbances, a first order time lag (adjustable time constant: DER_N) is integrated in the algorithm for forming the D action. Generally, a small value ($DER_N < 8$) is adequate for DER_N to achieve a successful outcome. If $DER_N > 2/CYCLE$ is configured, the time lag is disabled.

The step response in the time domain is as follows:

$$MV(t) = KP * ER(t) * (1 + \int dt / TI + DER_N * TD * EXP(t * DER_N))$$

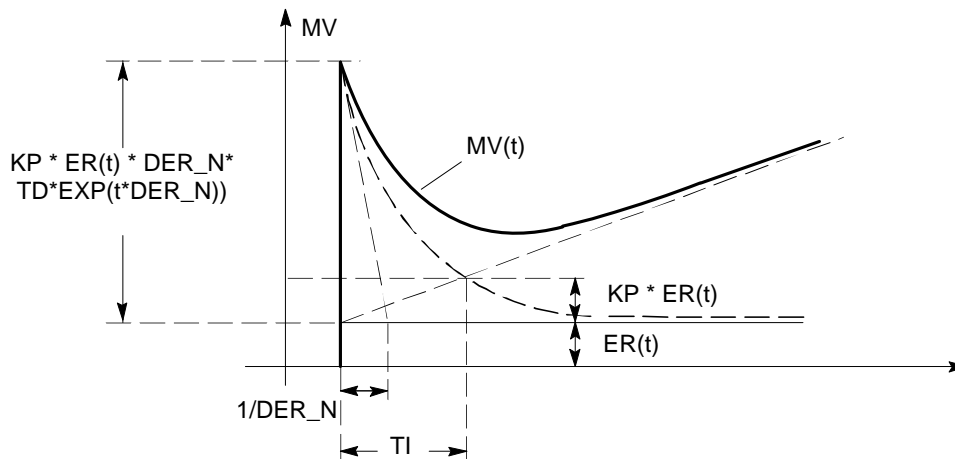


Figure 11-23 Step Response of the PID Controller

3.13.13 Using and Assigning Parameters to the PID Controller

The PI/PID functions of the Standard PID Control are capable of controlling most processes in industry. Functions and methods beyond the scope of this controller are only necessary in special situations.

One practical problem nevertheless remains the assignment of parameters to PI/PID controllers, in other words finding the “right” settings for the controller parameters. The quality of the parameter assignment is the decisive factor in the quality of the PID control and demands either considerable practical experience, specialist knowledge or a lot of time.

3.13.14 Permitted Ranges for TI and CYCLE

Due to the limited accuracy of the REAL numbers calculated in the CPU, the following effect can occur during integration: If the sampling time CYCLE is too small compared with the reset time TI and if the input value ER of the integrator is too small compared with its output value I, the integrator does not respond and remains at its current output value.

This effect can be avoided by remembering the following rule when assigning parameters:

$$CYCLE > 10^{-4} * TI$$

With this setting, the integrator reacts to changes in the input values that are in the range of ten millionths of a percent of the current output value:

$$ER > 10^{-10} * I$$

To ensure that the transfer function of the integrator algorithm corresponds to the analog response, the sampling time should be less than 20% of the reset time TI, in other words TI should be five times higher than the selected sampling time:

$$CYCLE < 0.2 * TI$$

The algorithm permits values for the sampling time up to $CYCLE \leq 0.5 * TI$.

3.13.15 Permitted Ranges for TD and CYCLE

To allow the derivative unit to process its calculation algorithm correctly in the CPU, keep to the following rules when assigning the time constants:

$$TD \geq CYCLE \text{ and}$$

$$1 / DER_N \geq 0.5 * CYCLE$$

If a value less than CYCLE is set, the derivative unit operates as if TD had the same value as CYCLE.

If 1/DER_N is set to a value $< 0.5 * CYCLE$, the derivative unit operates without a delay. The input step change is then multiplied by the factor TD/CYCLE and this value is applied to the output as a “needle pulse”. This means that in the next processing cycle, D is reset to zero.

3.13.16 Windup

Although many aspects of a control system can be understood based on linear theory, some nonlinear effects must be accounted for in practically all controllers. Windup is such a phenomenon, which is caused by the interaction of integral action and saturations. All actuators have limitations: a motor has limited speed, a valve cannot be more than fully opened or fully closed, etc. For a control system with a wide range of operating conditions, it may happen that the control variable reaches the actuator limits.

When this happens the feedback loop is broken and the system runs as an open loop because the actuator will remain at its limit independently of the process output. If a controller with integrating action is used, the error will continue to be integrated. This means that the integral term may become very large or, colloquially, it “winds up”. It is then required that the error has opposite sign for a long period before things return to normal. The consequence is that any controller with integral action may give large transients when the actuator saturates. The standard PID has an internal mechanism in order to prevent controller wind up called Anti-Windup and is enabled by default.

3.14 PWM Signal Generator (PWM_GEN)

Table 11-28 PWM_GEN instruction

LAD/FBD	Description
	<p>The pulse generator module transforms the input variable INV modulating the pulse width into a pulse sequence with a period time, which has to be configured in PERIOD.</p> <p>The duration of a pulse per period is proportional to the input value. The cycle set by RUN is not identical to the processing cycle of the pulse generator.</p>

Supported Properties: None

Table 11-29 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
INV	Real	Input variable as duty cycle. $0 \leq INV \leq 100$
PERIOD	Time	Period of PWM signal
Q	Bool	Pulse output

Application

The PWM generation function generates the pulse output of a continuous controller so that proportional actuators can be controlled by pulses using the Standard PID Control. This allows PID two-step and three-step controllers to be implemented with pulse width modulation.

3.15 PID Tuner by Relay Method (RELAY_TUNE)

Table 11-30 RELAY_TUNE instruction

LAD/ FBD	Description
	<p>The PID autotuner works by performing a frequency-response estimation experiment. It injects test signals into the plant and tune PID gains based on an estimated frequency response based on Relay (Åström–Hägglund) method.</p>

Supported Properties: None

Table 11-31 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
PV	Real	Process variable
SV	Real	Setpoint value. The process variable (PV) will be oscillated around the SP during the tuning process
TUN_MOD	INT	Tune mode. 0 = P, 1 = PI, 2 = PID
RSP_MOD	INT	Response mode. 0 = Normal, 1 = Conservative, 2 = Aggressive
CYCLE	Time	Sample time
MV	Real	Manipulated value
PHASE	Int	Current operating phase
QBUSY	Bool	1 = Identifying, 0 = Process accomplished
KP	Real	Proportional gain
TI	Real	Reset time
TD	Real	Derivative time
CP0	Real	Control parameter 0
CP1	Real	Control parameter 1

Static Members		
MV_LO	Real	Low limit of mv
MV_HI	Real	High limit of mv
HYST	Real	Hysteresis (%)

Application

The performance of an automatic PID controller tuning method based on relay feedback is studied in the presence of deterministic disturbances. It is found that the occurrence of any static load disturbance could cause significant errors in the estimates of the ultimate gain and period. However, the resultant asymmetry of the relay switching intervals can be used as an error indicator, or used to compute a self-corrective bias to restore accuracy of the estimates. This corrective bias is found to be functional even in the presence of moderate nonlinearity. A reliable self-biasing auto-tuner is thus resultant. The effect of a less common sinusoidal load could be more serious since it may not be detectable and hence more prior knowledge about its presence is required.

You should use this tuner for applications that have a large time lag and large time consuming for using step response tuners (see SELF_TUNE). For example, a furnace with at least one hour time lag step response should be tuned by RELAY_TUNE.

When you change the RUN input from False to True, the tuning process will be started. It will be proceeded in several phases and finally will be accomplished at phase 6. You can see current state of tuner by checking the PHASE output. In phase 6 the tuner has been accomplished its internal examinations and generates PID operation gains KP, TI, TD, CP0 and CP1 respected the TUN_MOD and RSP_MOD inputs. You can get PID parameters for each operation mode while the tuner remains in phase 6. In the phase 6 if you change the RUN input from True to False, its internal state will be reset.

3.16 Ramp Soak (RMP_GEN)

Table 11-32 RMP_GEN instruction

LAD/ FBD	Description
	<p>The ramp soak RMP_GEN supplies the output variable OUT according to a defined schedule. This function is started by setting the input bit RUN. If the value for operation mode OP_MOD = True, the function is started again at the first time slice outv[0] after the last time slice outv[TMV_OUT_N] has been output. There is no interpolation between the last and first time slice when operation mode is in cyclic repetition.</p> <p>The sequence of the ramp soak is defined by specifying a series of time slices (between coordinates) in a shared array of user data type (UDT) with the time values TMV_OUT_S[i].TMV and the corresponding output values TMV_OUT_S [i].OUTV.</p> <p>TMV_OUT_S [i].TMV specifies the length of time of the time slices. There is linear interpolation between the coordinates.</p>

Supported Properties: None

Table 11-33 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
DF_OUT	Real	Default output variable
DF_OUT_ON	Bool	Output default value on
TMS_N	Int	Time slice number
TM_CONT	Time	Time to continue
CONT_ON	Bool	Continue
HOLD	Bool	Hold output variable
OP_MOD	Int	Operation mode. 0 = Single run, 1 = Continue last value, 2 = Repeat all
RST_ON	Bool	Restart
CYCLE	Time	Sample time
TMV_OUT_N	Int	Number of coordinates
TMV_OUT_S	Variant[*]	Coordinates source is an array of structure. The structure must contains 'TMV' & 'OUT' elements
OUT	Real	Output variable
TGT_OUT	Real	Target output variable
QRMP_OP	Bool	Ramp operating
RMP_DIR	Int	Ramp direction. 0 = Disabled, 1 = Increasing, 2 = Soaking, 3 = Decreasing
N_ATMS	Int	Number of acting time slice
PROG	Real	Progress (%)
RS_TM	Time	Residual slice time
T_TM	Time	Total time
RT_TM	Time	Residual total time

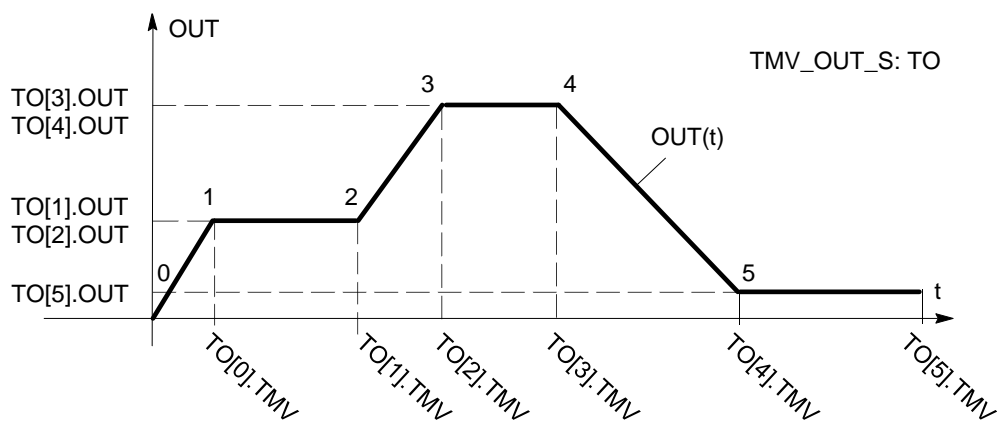


Figure 11-24 Ramp Soak with Start Point and Six Time Slices

TIP

With n time slices the time value TMV_OUT_S [n-1].TMV for the last time slice will be processed. The processing time of a ramp soak is calculated from the initial value down to 0.

TIP

During the interpolation of the ramp soak between the time slices, the output value may pause occasionally if the sampling time CYCLE is very small compared with the time between the time slices TMV_OUT_S [n].TMV. The ramp soak cannot produce flat linear forms arbitrarily because of the computational accuracy of the CPU. If the ramp soak is too flat, the output value will pause at the respective time slice for a while and then integrates with the minimum gradient to the next time slice.

Remedy: Reduce the time between the time slices by inserting additional time slices. This way you will get the ramp soak output closer to the desired flat ramp soak in a trapeze from.

3.16.1 Using the Ramp Soak

The time slice parameters `TMV_OUT_N`, `TMV_OUT_S [i].TMV` and `TMV_OUT_S [i].OUT` are located in an array of an user data type (UDT).

- The parameter `TMV_OUT_S [i].TMV` must be specified in the Time format.
- The parameter `TMV_OUT_S [i].OUT` must be specified in the Real format.
- The way in which the coordinates and time slices are counted is illustrated in the following diagram.

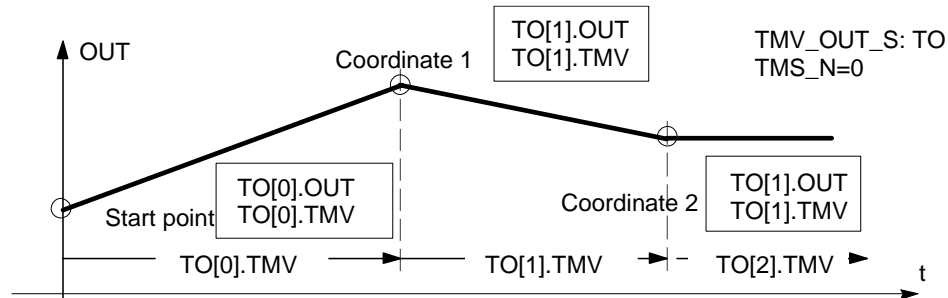


Figure 11-25 Counting the Coordinates and Time Slices

In normal operation, the ramp soak interpolates according to the following function where $0 \leq n \leq (\text{TMV_OUT_N} - 1)$.

3.16.2 Configuring the Ramp Soak

The number of configured coordinates (`TMV_OUT_N`) and the values for the setpoint `SP` assigned to the individual time slices can be monitored and are located in an array of user data type. The output of the ramp soak begins at start point `[0]` and ends with the coordinate `[TMV_OUT_N]`.

3.16.3 Modes of the Ramp Soak

By influencing the control inputs, the following ramp soak statuses and operating modes can be implemented:

- 1- Ramp soak on for a single run.
- 2- Default value at output of ramp soak.
- 3- Repetition on (cyclic mode).
- 4- Hold processing of the ramp soak (hold setpoint value).
- 5- Set the time slice and time to continue (the remaining time `TM_CONT` and the time slice number `TMS_N` are redefined).
- 6- Update the total processing time and total time remaining.

3.16.4 Activating the Ramp Soak

The change in `RUN` from `False` to `True` activates the ramp soak. After reaching the last time slice, the ramp soak (curve) is completed. If you want to restart the function manually, `RUN` must first be set to `False` then back to `True`.

During a complete restart, the `OUT` output is reset to 0.0 and the total time or total remaining time is calculated. When it changes to normal operation, the ramp soak is processed immediately from the start point according to the selected mode. If you do not require this, the parameter `RUN` when the complete restart must be set to `False`.

WARNING

The function block does not check whether an array with the length `TMV_OUT_N` exists or not and whether the parameter `TMV_OUT_N` number of time slices matches the array length. If the parameter assignment is incorrect, the CPU changes to `STOP` due to an internal runtime error.

3.16.5 Preassigning the Output, Starting the Traveling Curve

If `DF_OUT_ON = True`, the output value of the ramp soak is set to the signal value `DF_OUT`. If `DF_OUT_ON = False`, the curve starts from this point.

TIP

The switch `DF_OUT_ON` only has an effect when the ramp soak is activated (`RUN = True`).

The changeover from `DF_OUT_ON = False` is followed by the linear adjustment of `OUT` from the selected setpoint to the output value of the current time slice number `TMV_OUT_S[N_ATMS].OUT`.

Internal time processing is continued even when a fixed setpoint is applied to the output (`RUN = True` and `DF_OUT_ON = True`).

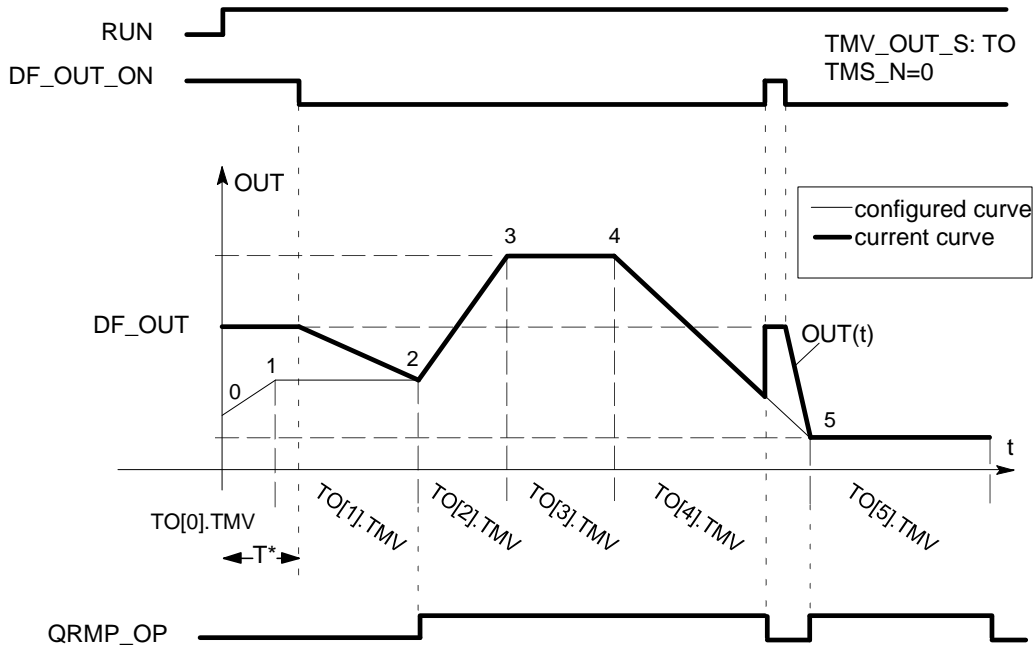


Figure 11-26 Influencing the Ramp Soak with the Default Signal `DF_OUT_ON`

When the ramp soak is started with `RUN = True`, the fixed setpoint `DF_OUT` is output until `DF_OUT_ON` changes from `True` to `False` after the time T^* . At this point, the time `TO[0].TMV` and part of the time `TO[1].TMV` has expired. The output value `OUT` is moved from `DF_OUT` to `TO[2].OUT`.

The configured curve is only output starting at coordinate 2, where the output signal `QRMP_OP` changes to the value `True`. When the preassigned signal `DF_OUT_ON` changes from `False` to `True` while the travel curve is being executed, the output value `OUT` jumps without delay to `DF_OUT`.

3.16.6 Cyclic Mode On

If the cyclic repetition mode is turned on (`OP_MOD=2`), the ramp soak returns to the start point automatically after outputting the last time slice value and begins a new cycle.

There is no interpolation between the last time slice and the start point. The following must apply to achieve a smooth transition: `TMV_OUT_S[TMV_OUT_N-1].OUT = TMV_OUT_S [0].OUT`.

3.16.7 Hold Setpoint Value

With `HOLD = True`, the value of the output variable (including the time processing) is frozen. When this is reset (`HOLD = False`), the ramp soak continues at the point of interruption `TMV_OUT_S[x].TMV`.

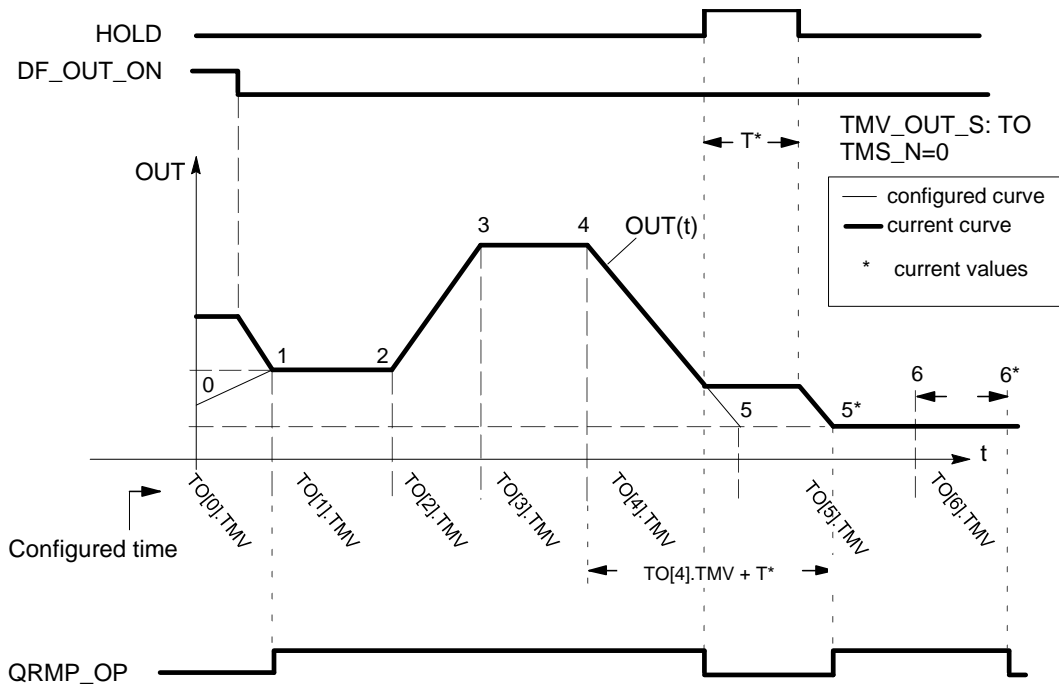


Figure 11-27 The Effect of the Hold Signal HOLD on the Ramp Soak

The processing time of the ramp soak is increased by the hold time T^* . The ramp soak follows the configured curve from the time slice to the signal change for HOLD (False → True) and from time slice 5* to time slice 6*, in other words the output signal QRMP_OP has the value True.

3.16.8 Selecting the Time Slice and Time to Continue

If the control input CONT_ON is set to True to continue processing, then processing continues at the time TM_CONT with the time slice TMS_N. The time parameter TM_CONT determines the time remaining that the ramp soak requires until it reaches the destination time slice TMS_N.

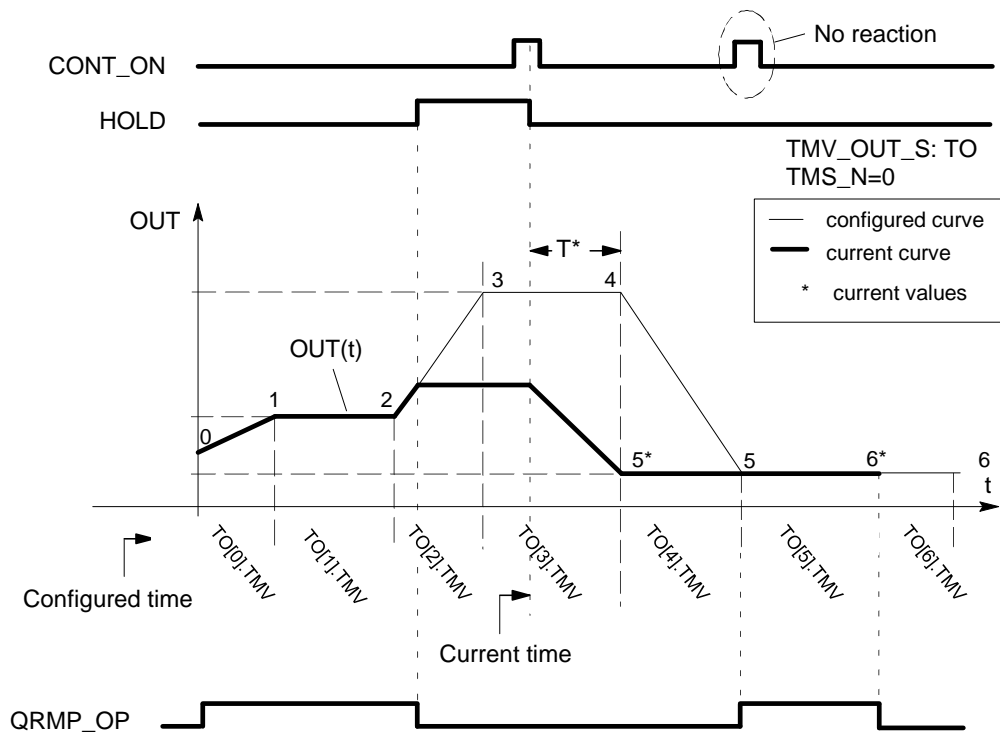


Figure 11-28 How the HOLD Hold Signal and the CONT_ON Continue Signal Affect the Ramp Soak

The following applies to the example: If HOLD = True and CONT_ON = True and if the following is selected time slice number to continue TMS_N = 5 and time remaining to selected time slice TM_CONT = T* then the configured coordinates 3 and 4 are omitted in the processing cycle of the ramp soak. After a signal change at HOLD from True to False the curve only returns to the configured curve starting at coordinate 5.

The output QRMP_OP is only set when the ramp soak has worked through the curve configured by the user.

3.16.9 Updating the Total Time and Total Time Remaining

In every cycle, the current time slice number N_ATMS, the current time remaining until the time slice RS_TM is reached, the total time T_TM and the total time remaining until the end of the ramp soak RT_TM is reached are updated.

If there are on-line changes to TMV_OUT_S[n].TMV, the total time and the total time remaining are changed. Since the calculation of T_TM and RT_TM greatly increases the run time of the function block if there are a lot of time slices, the calculation is only performed after a complete restart or when RST_ON = True. The time slices TMV_OUT_S [0 to TMV_OUT_S-1].TMV between the individual coordinates are totalled and indicated at the output for the total time T_TM and for the total remaining time RT_TM.

Please remember that the calculation of the total times requires a relatively large amount of CPU time.

3.17 Limiting the Rate of Change of a Value (ROC_GEN)

Table 11-34 ROC_GEN instruction

LAD/ FBD	Description
	<p>The ROC_GEN function limits the rate of change of the setpoints processed in the controller separately for the rate of change up and rate of change down and also separately for the positive and negative ranges.</p> <p>The limits for the rate of change of the ramp function in the positive and negative range of the reference variable are entered at the four inputs UPRLM_P, DNRLM_P, UPRLM_N and DNRLM_N. The rate of change is an up or down rate per second. Faster rates of change in the setpoint are delayed by these limits.</p> <p>If, for example, UPRLM_P is configured to 10.0 [technical range of values/s], the following values are added to the 'old value' of OUT in each sampling cycle as long as $inv > OUT$:</p> <p>Sample time $1\text{ s} \rightarrow OUT\text{ old} + 10$ $100\text{ ms} \rightarrow OUT\text{ old} + 1$ $10\text{ ms} \rightarrow OUT\text{ old} + 0.1$</p> <p>How signals are handled by the function is illustrated by the following figure based on an example. Step functions at the input INV(t) become ramp functions at output OUT(t).</p>

Supported Properties: None

Table 11-35 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
INV	Real	Input variable
UPRLM_P	Real	Up rate limit in positive range
DNRLM_P	Real	Down rate limit in positive range
UPRLM_N	Real	Up rate limit in negative range
DNRLM_N	Real	Down rate limit in negative range
H_LM	Real	Input variable high limit
L_LM	Real	Input variable low limit
CYCLE	Time	Sample time
OUT	Real	Output variable
QH_LM	Bool	Output variable high limit
QL_LM	Bool	Output variable low limit

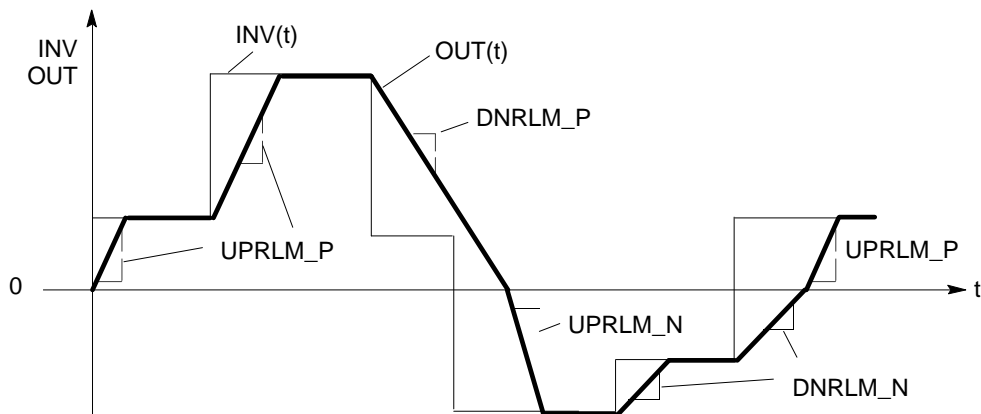


Figure 11-29 Limiting the Rate of Change of the Input Variable INV(t)

No signal is output when the rate of change limits are reached.

3.18 Scale (SCALE)

Table 11-36 SCALE instruction

LAD/ FBD	Description
	<p>Scales the normalized real parameter INV where (0.0 <= INV <= 1.0) in the data type and value range specified by the OUT_L and OUT_H parameters:</p> $OUT = INV (OUT_H - OUT_L) + OUT_L$

Supported Properties: None

Table 11-37 Data types for the parameters

Parameter	Data type	Description
INV	AnyNum	Input variable
OUT_L	AnyNum	Output low limit
OUT_H	AnyNum	Output high limit
OUT	AnyNum	Scaled output

3.19 Gain Scheduling (SCH_GEN)

Table 11-38 SCALE instruction

LAD/ FBD	Description
	<p>The SCH_GEN generates a signal by using an array of key value pair and maps the change in input variable INV to output OUT by the schedule table SCHD_S.</p>

Supported Properties: None

Table 11-39 Data types for the parameters

Parameter	Data type	Description
INV	Real	Input variable
KV_N	Int	Number of key value pairs. $2 \leq x \leq 8$
SCHD_S	Real[*,*]	Schedules list source. Key value pairs of (INV,OUT)
OUT	Real	Output variable

Application

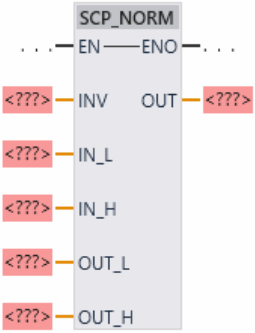
Gain scheduling is valuable to adjust the controller parameters to different operating points, if a nonlinear process shows a different behavior for each of its operating points.

The schedule for the adaptation of the parameters is determined by separate experiments at each of the different operating points and is stored in the SCH_GEN function block. During a change between operating points the controller automatically retrieves the correct parameters in the schedule. A new identification of the process behavior according to new measurement data is not necessary, in contrast to a fully adaptive controller.

However, gain scheduling can be applied only if the nonlinearity of the process can be traced back to only one measurable process variable in a reproducible way.

3.20 Scale With Parameters (SCP_NORM)

Table 11-40 SCP_NORM instruction

LAD/ FBD	Description
	<p>The instruction will take input, use input minimum IN_L and maximum IN_H parameters as well as outputs minimum OUT_L and maximum OUT_L parameters and convert the output scaling based on them.</p> <p>This function commonly used for working with analog signals</p>

Supported Properties: None

Table 11-41 Data types for the parameters

Parameter	Data type	Description
INV	AnyNum	Input variable
IN_L	AnyNum	Input low limit
IN_H	AnyNum	Input high limit
OUT_L	AnyNum	Output low limit
OUT_H	AnyNum	Output high limit
OUT	AnyNum	Scaled output

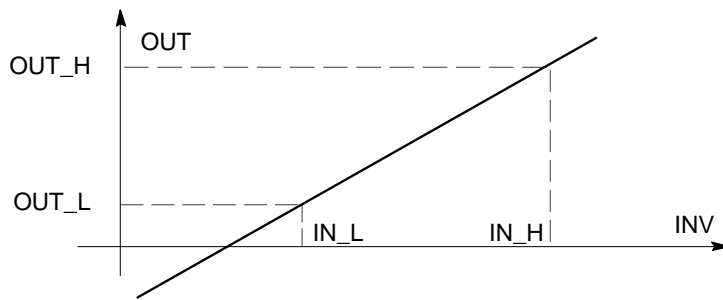


Figure 11-30 SCP_NORM normalization curve

NOTICE

The function does not limit any values and the parameters are not checked. If you enter the same value for IN_L and IN_H, division by zero can occur in the function. The function does not rectify this fault.

3.21 PID Self Tuner (SELF_TUNE)

Table 11-42 SELF_TUNE instruction

LAD/ FBD	Description
	<p>One tuning method presented by Ziegler and Nichols is based on a process information in the form of the open-loop step response obtained from a bump test. This method can be viewed as a traditional method based on modeling and control where a very simple process model is used. The step response is characterized by its parameters and the controller parameters are then obtained from the characterized process parameters.</p>

Supported Properties: None

Table 11-43 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
PV	Real	Process variable
TUN_MOD	Int	Tune mode. 0=P, 1=PI, 2=PID
RSP_MOD	Int	Response mode. 0=Normal, 1=Conservative, 2=Aggressive
CYCLE	Time	Sample time
ACCUTUNE	Bool	Tune accurately and more robust but more time consuming
MV	Real	Manipulated value

PHASE	Int	Current operating phase
QBUSY	Bool	1 = Identifying, 0 = Process accomplished
DELAY	Time	Time lag include any existing dead time
RANK	Int	Controllability rank. $x > 100$: Easy, $30 < x < 100$: Somewhat Easy, $x < 30$: Difficult
KP	Real	Proportional gain
TI	Real	Reset time
TD	Real	Derivative time
CP0	Real	Control parameter 0
CP1	Real	Control parameter 1
Static Members		
MV_STP	Real	Manipulated value step
MAN_MV	Real	Initial manual power
WRM_TM	Time	Warmup time
WRM_MVT	Time	Manual power startup time
WRM_INT	Time	Warmup evaluation interval
MAN_INT	Time	Manual evaluation interval
STD_TH	Real	Steady state threshold
STRT_TH	Real	Start state threshold
CONT_STP	Int	Continue tuning steps count
CUR_STP	Int	Current tuning step

3.21.1 Area of Application

PID SELF_TUNE is particularly useful for the following:

- Temperature controls (main application)
- Level controls
- Flow controls

In flow controls, a distinction must be made between situations in which only the control valve itself must be controlled and situations in which the control valve regulates a process involving a time lag. The PID SELF_TUNE cannot be used for simple control of a valve.

3.21.2 Process Requirements

The process must meet the following requirements:

- Stable, time lag, asymptotic transient response
- Time lags not too large
- Adequate linear response with an adequately large operating range
- Process controllable with a monopolar actuating signal 0 to 100%
- Little disturbance in temperature processes
- Adequate quality of the measured signals in the sense of an adequately high signal-to-noise ratio.
- Process gain not too high

3.21.3 Transient Response

The process must have a stable, asymptotic transient response with time lag.

After a step change in the manipulated variable (MV) the process variable must change to a steady state as shown in following figure. This therefore excludes processes that have an oscillating response without control and processes that are not self-regulating (integrator in the process).

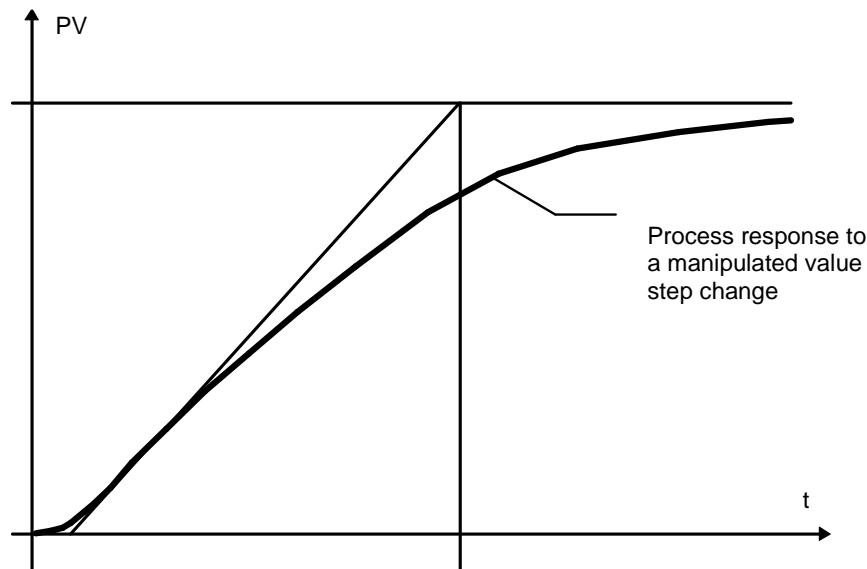


Figure 11-31 Process Response

3.21.4 Time Lags

The process must not involve large time lags. The range of application can be specified. The time lag includes any existing dead time. Most temperature processes are within the default configurations of the function block both a PI or a PID controller can be designed for this range.

3.21.5 Linearity and Operating Range

The process must have an adequately linear response over an adequately large operating range. This means that both during identification and during normal controlled operation, non-linear effects within the operating range can be ignored. It is, however possible to re-identify the process when the operating point changes if the adaptive process is repeated in the close vicinity of the new operating point and providing that the non-linearity does not occur during the adaptation.

If certain static non-linearities (for example valve characteristics) are known, it is always advisable to compensate these with a ramp soak to linearize the process behavior or use gain scheduling method in order to set suitable PID parameters for each individual working area of process.

3.21.6 Monopolar Actuating Signal

It must be possible to control the process with a monopolar actuating signal. Processes requiring active heating and active cooling for temperature control cannot currently be optimized with the PID Self-Tuner.

3.21.7 Disturbances in Temperature Processes

Disturbances such as thermal transfer to neighboring zones or heating or cooling due to changes in the equipment status must not affect the overall temperature process to any great extent. In some circumstances, adaptation at the operating point is necessary.

3.21.8 Quality of the Measured Signals

The quality of the measured signals must be adequate, in other words the signal-to-noise ratio must be high enough.

3.21.9 Process Gain

The process gain must not be too high.

Normalization of the process values is not required. The process gain K can, in some circumstances, include physical units, for example:

$$K = \frac{\Delta PV}{\Delta MV}, [K] = \frac{^{\circ}\text{C}}{\%}$$

The final controller design is based on a calculation of the process gain K and can therefore, in principle, compensate any values of K. During the learning phase, however, K is initially unknown and with extreme combinations of gain and test step change, overshoot cannot be avoided.

3.21.10 Processes with a Control Valve with Integral Action

In processes with control valves with an integral action, there are further requirements in addition to those above:

The motor actuating time of the control valve must be less than the time required to find a point of inflection following a step change in the manipulated value.

If this is not the case, the process involved is often a flow control in which only the control valve is effective as the dominating process action. The use of the PID Self-Tuner is then not advisable. You can set the PI step controller according to the following rule of thumb:

GAIN (KP) = 1, TI = control valve actuating time

3.21.11 Learning Phases

The learning process involves the following steps:

- PHASE = 0:
When an instance of SELF_TUNE FB is created or the tuner is disabled, the parameter PHASE has the default zero.
- PHASE = 1-5:
Tuning transition steps
- PHASE = 6:
In this phase, the tuner generates optimized parameters relative to the TUN_MOD and RSP_MOD. You can get PID parameters for each operation mode while the tuner remains in phase 6. In the phase 6 if you change the RUN input from True to False, its internal state will be reset and the tuner will go to PHASE 0.

NOTICE

Before activating RUN, the process must be in a steady state otherwise, you must then wait until the process variable remains constant. This achieves a steady state ("cold" process state, initial state).

TIP

If it takes an extremely long time until the steady state is reached (creeping transient response in temperature processes) you can lower the steady state threshold by increasing 5% the STD_TH value until pass the current tuning phase.

TIP

If the process cannot start by suddenly change in MV from cold state, you should set a value greater than T#0s for WRM_TM static member to enable the warmup process. For example, some furnaces need a warmup process in order to be applicable otherwise they may damage due to sudden temperature change.

TIP

If the process is not linear or you want to run a gain scheduling program for PID parameters, you can set a value greater than 0 for CONT_STP static member. In this case the controller repeats learning phases for a number of steps determined by CONT_STP. For example, if you want to get parameters for a nonlinear furnace you can set a value 3 for CONT_STP so, the learning phases will be repeated 3 times. Every repeat learning phase will be increasing the MV by the value specified by MV_STP. When a learning phase accomplishes, the QBUSY is set to false. You can check this flag in order to realize that the current learning phase has been finished and pick the generated parameters by self-tuner. In the next cycle the QBUSY will be set in order to show a new learning phase has been started.

3.22 Extracting the Square Root Normalization (SQRT_NORM)

Table 11-44 SQRT_NORM instruction

LAD/ FBD	Description
	<p>If the input variable supplied by a sensor is a physical value that is in a quadratic relationship to the measured input variable, the changes in the input variable must first be linearized before they can be processed further in the other signal conditioners.</p>

Supported Properties: None

Table 11-45 Data types for the parameters

Parameter	Data type	Description
INV	AnyNum	Input variable
SQRT_HR	AnyNum	Normalization high range
SQRT_LR	AnyNum	Normalization low range
OUT	AnyNum	Normalized output

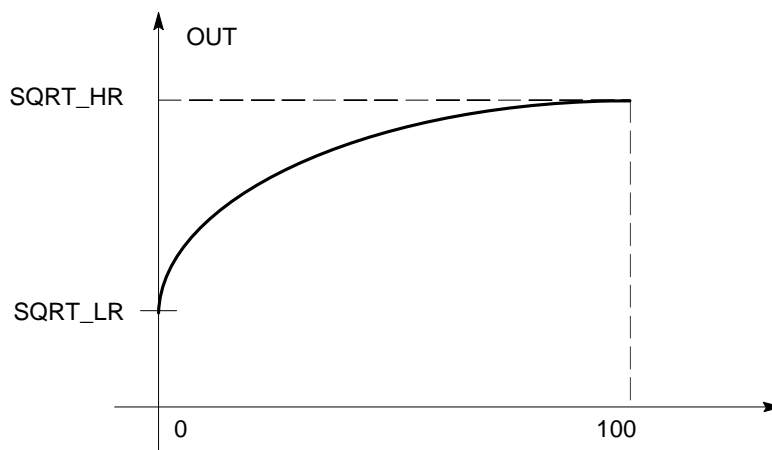


Figure 11-32 The square root normalization

3.23 Stack Collection (STACK)

Table 11-46 STACK instruction

LAD/ FBD	Description
	<p>A stack is an array structure of function calls and parameters used in modern computer programming and CPU architecture. Elements in a stack are added or removed from the top of the stack, in a “last in first, first out” or LIFO order.</p>

Supported Properties: None

Table 11-47 Data types for the parameters

Parameter	Data type	Description
N	Int	Maximum depth after reset
PUSH	Bool	Basic stack operations
POP	Bool	Basic stack operations
R1	Bool	Over-riding reset
IN	Variant	Input to be queued
OUT	Variant	First element data
BUFFER	Variant[*]	External array
EMPTY	Bool	Stack empty
OFLO	Bool	Stack overflow

3.24 Three Step Signal Generator (THREE_STEP_GEN)

Table 11-48 THREE_STEP_GEN instruction

LAD/ FBD	Description
	<p>Three step generator for PIDs when both the direct and inverse control works together. In the “three-step generator” mode, the actuating signal can have three states, for example depending on the actuator and process: more – off – less, forwards – stop – backwards, heat – off – cool etc.</p>

Supported Properties: None

Table 11-49 Data types for the parameters

Parameter	Data type	Description
INV1	Real	Input variable 1
INV2	Real	Input variable 2
COEFF1	Real	Input variable 1 coefficient
COEFF2	Real	Input variable 2 coefficient
DEADB_W	Real	Dead band width
FALS_DB	Bool	Falsify error outside dead band. 0=Disabled, 1=Enabled
OUT1	Real	Output 1
OUT2	Real	Output 2
QOUT1DB	Bool	INV1 is within dead band
QOUT2DB	Bool	INV2 is within dead band

3.25 Weighing System (WEIGH)

Table 11-50 WEIGH instruction

LAD/ FBD	Description
	Complete weighing system with Zero, Tare and calibration functions.

Supported Properties: None

Table 11-51 Data types for the parameters

Parameter	Data type	Description
GRS_WT	Real	Raw value from loadcell (gross weight)
TARE	Bool	Command for Tare when the system is calibrated
ZERO	Bool	Command for zero when the calibration starts
CALIB	Bool	Command for calibration when the calibration finishes
REF_WT	Real	Reference weight when the calibration finishes
OUT	Real	Scaled output

12

Technology Instructions

Throughout a development process of control systems, control engineers deal with challenges like shorter development time, higher quality and flexibility requirements and reusability of the control code. Since existing technologies and approaches are limited by their effectiveness, new approaches are needed. This chapter will provide common but high-level application solutions in order to lowering the programming time for PLC developers.

 TIP

Some technology instructions may need a valid license to be compiled. In order to activate these instructions, you should install license files on your programming device. To obtain a license file, you must contact the INTELART's support unit.

1. Temperature Control

Temperature Controllers control temperature so that the process variable will be the same as the set point, but the response will differ due to the characteristics of the controlled object and the configurations of the Temperature Controller. Typically, a response where the set point is reached as quick as possible without overshooting, is required in a Temperature Controller. There are also cases where a response quickly increases the temperature even if it overshoots is required or where a response slowly increases the temperature is required.

1.1 Temperature Control by TEMP_CONTROLLER

Table 12-1 TEMP_CONTROLLER instruction

LAD/ FBD	Description
	<p>Temperature control is a process in which change of temperature of a space, or of a substance, is measured or otherwise detected, and the passage of heat energy into or out of the space or substance is adjusted to achieve a desired temperature. TEMP_CONTROLLER provides a complete function block optimized for temperature control with algorithms for loading and storing patterns and programs.</p>

Supported Properties: None

Table 12-2 Data types for the parameters

Parameter	Data type	Description
PV	Real	Process variable
MIN_SP	Real	Minimum allowed value for setpoints
MAX_SP	Real	Maximum allowed value for setpoints
MAN_MV	Real	Manual manipulated variable (%)
MAN_SP	Real	Manual setpoint value

CYCLE	Time	Sample time of controller
HOLD	Bool	Hold current setpoint
STARTUP	Bool	Initialize controller on startup
OPTIM_ON	Bool	Optimizer enable. 0=Disabled, 1=Enabled
MAN_ON	Bool	Manual value on (variable MAN_MV)
MAN_SP_ON	Bool	Manual setpoint on. 0=Ramp/Soak, 1=Step
ACK_ALM	Bool	Acknowledge alarm
PTRN_CNT	Int[*]	Program patterns table count
PTRN_TMT	Time[*,*]	Program patterns time table
PTRN_SPT	Real[*,*]	Program patterns setpoint table
PRG_TABLE	Variant[*]	Program table
SELF_TUNER	SELF_TUNE	Self-tuner instance (Optional)
OPTIMIZER	TEMP_OPT	Optimizer instance (Optional)
MV	Real	Manipulated variable (%)
TGT_SP	Real	Target setpoint
CUR_SP	Real	Current setpoint
PROG	Real	Progress (%)
CON_DEV	Real	Control deviation (%)
RS_TM	Time	Residual slice time
T_TM	Time	Total elapsed time
RT_TM	Time	Residual total time
TUN_DELAY	Time	Delay of system response
ALARM	Int	Alarm. <ul style="list-style-type: none"> • 1=Warmup timeout • 2=Temperature under range • 3=Temperature over range • 4=Invalid program table • 5=Invalid time patterns • 6=Invalid setpoint patterns • 7=Invalid tuner instance • 8=tune while control impossible • 9=null optimizer • 10=No program found
RMP_DIR	Int	Ramp direction. <ul style="list-style-type: none"> • 0 = Disabled • 1 = Increasing • 2 = Soaking • 3 = Decreasing
N_ATMS	Int	Number of acting time slice
TUN_PHASE	Int	Current tunning operation phase
TUN_RANK	Int	Rank of system controllability. <ul style="list-style-type: none"> • $x < 30$: Weak • $30 < x < 100$: Moderate • $x > 100$: Good
QRMP_OP	Bool	Ramp operating
QCON_ON	Bool	Control process run
QPWR_ON	Bool	Main power switch
QTUNNING	Bool	Tunning is in process. True = Identifying False = Tune accomplished
Static Members		
PID_CONTROLLER	PID_STD	Internal PID Controller Instance
OPTIM_RMP	Real	Deviation suppressor threshold in ramp when optimizer enabled (%)
OPTIM_RMP_SPAN	Real	Ramp span when optimizer enabled (%)

OPTIM_STD_UP	Real	Deviation suppressor threshold in soak up when optimizer enabled (°C)
OPTIM_STD_DN	Real	Deviation suppressor threshold in soak down when optimizer enabled (°C)
WDOG_SPAN	Real	Watchdog Range (°C)
MAX_RMP_RATE	Real	Maximum rate of SP ramp (°C/M)
CUTOFF_TEMP	Real	Cutoff temperature
CON_WRM_TEMP	Real	Controller warmup temperature
CON_WRM_POWER	Real	Controller warmup end power
CON_WRM_TMV	Time	Controller warmup time value
CON_WRM_TMOUT	Time	Controller warmup timeout
TUN_WRM_TMV	Time	Tuner warmup time value
MAN_DS_PULUP_TRG	Time	Dev supp pull up time trigger in manual setpoint mode
MAN_DS_PULDN_TRG	Time	Dev supp pull down time trigger in manual setpoint mode
PWR_LOSS_ACT	Int	Power loss action. <ul style="list-style-type: none"> • 0=Stop • 1=Hold • 2=Restart • 3=Continue
START_MOD	Int	Operation start Mode. <ul style="list-style-type: none"> • 0=Normal • 1=Start from Current PV • 2=Start from current PV subtract warmup time
STOP_MOD	Int	Operation stop mode <ul style="list-style-type: none"> • 0 = Single run • 1 = Continue last value • 2 = Repeat all
CUTOFF_MOD	Int	Cutoff mode. <ul style="list-style-type: none"> • 0=Absolute • 1=Relative
CUR_PTRN	Int	Set current selected pattern
CON_MOD	Int	Control mode. <ul style="list-style-type: none"> • 0=P • 1=PI • 2=PID
TUN_RSP_MOD	Int	Response mode. <ul style="list-style-type: none"> • 0=Normal • 1=Conservative • 2=Aggressive
PRG_CNT	Int	Current loaded program count
CMD_START_CON	Bool	Start controller command
CMD_STOP_CON	Bool	Stop controller command
CMD_START_TUN	Bool	Start self-tuner command
CMD_STOP_TUN	Bool	Stop self-tuner command
CMD_SKIP_STEP	Bool	Skip to next program command
CMD_LOAD_PTRN	Bool	Loads a pattern to program table
CMD_STORE_PTRN	Bool	Stores program table to patterns source and ensures ramps rate be lower than 'MAX_RMP_RATE'
CMD_CLR_ALL	Bool	Clear all pattern program tables
SM_INIT	Bool	Smooth initialization. <ul style="list-style-type: none"> • False=Disabled • True=Enabled
CON_WRM_ON	Bool	Controller warmup enabled. <ul style="list-style-type: none"> • False=Disabled • True=Enabled
WDOG_ON	Bool	Watchdog enable.

		<ul style="list-style-type: none"> • False=Disabled • True=Enabled
ACCUTUNE	Bool	Tune accurately and more robust but more time consuming
QPRG_TABLE_UPD	Bool	Program table updated

1.2 Temperature Control Optimizer (TEMP_OPT)

Table 12-3 TEMP_OPT instruction

LAD/ FBD	Description
	Optimizer for ramp/soak temperature control by connecting a ramp generator and a PID controller.

Supported Properties: None

Table 12-4 Data types for the parameters

Parameter	Data type	Description
RUN	Bool	Run mode
PID_INST	Variant	An INTELART PID function block instance
RMP_INST	Variant	An INTELART Ramp generator function block instance
Static Members		
STD_PULUP_DEV	Real	Deviation suppressor Pull up threshold in steady state
STD_PULDN_DEV	Real	Deviation suppressor Pull down threshold in steady state
INC_RMP_SPN	Real	Increasing ramp span. $0 < x < 100$
INC_PULUP_DEV	Real	Deviation suppressor Pull up threshold (%) in increasing state
INC_PULDN_DEV	Real	Deviation suppressor Pull down threshold in increasing state
INC_HOLD_G	Real	Gain when an increasing ramp process holds
INC_RST_G	Real	Resting gain in increasing state
DEC_RMP_SPN	Real	Decreasing ramp span. $0 < x < 100$
DEC_PULUP_DEV	Real	Deviation suppressor Pull up threshold in decreasing state
DEC_PULDN_DEV	Real	Deviation suppressor Pull down threshold in decreasing state
DEC_HOLD_G	Real	Gain when an decreasing ramp process holds
DEC_RST_G	Real	Resting gain in decreasing state
APP_RT	Real	Approaching rate. $0 < x < 1$
RST_RT	Real	Resting rate. $0 < x < 1$ (calculates automatically on start)
ADAPT_ON	Bool	Adaptive optimization on
STD_ON	Bool	Enable deviation suppressor in steady state
STD_PCT	Bool	Enable calc on percentage of error in steady state
INC_ON	Bool	Enable deviation suppressor in increasing state
INC_PCT	Bool	Enable calc on percentage of error in increasing state
DEC_ON	Bool	Enable deviation suppressor in decreasing state
DEC_PCT	Bool	Enable calc on percentage of error in decreasing state

13

Online and Diagnostic Tools

The "Online & diagnostics" shows the diagnostic status and tools of the device.

1. Status LEDs

The CPU and the I/O modules use LEDs to provide information about either the operational status of the module or the I/O.

1.1 Status LEDs on a CPU

The CPU provides the following status indicators:

POWER

- Solid green indicates device has been powered up

STOP/RUN

- Off indicates STOP mode
- Solid green indicates RUN mode
- Flashing indicates that the CPU is in TRANSIENT-TO-RUN or TRANSIENT-TO-STOP mode

ERROR

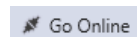
- Solid red indicates an error, such as an internal error in the CPU or a configuration error (mismatched modules)

2. Going online and connecting to a CPU

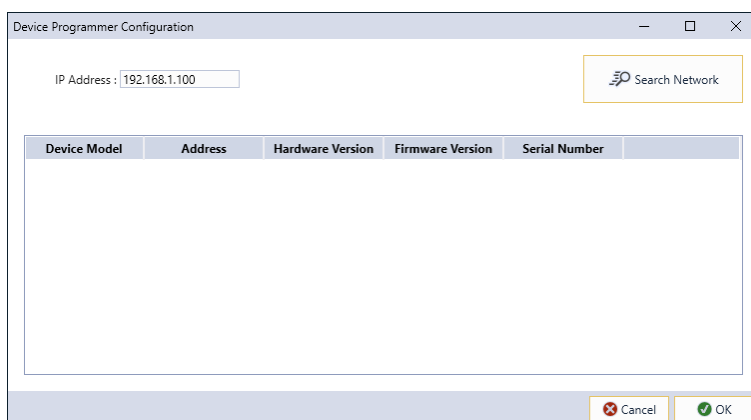
An online connection between the programming device and CPU is required for loading programs and project engineering data as well as for activities such as the following:

- Testing user programs
- Displaying and changing the operating mode of the CPU
- Displaying and setting the date and time of day of the CPU
- Displaying the module information
- Downloading user program
- Displaying diagnostics data
- Using a watch table to test the user program by monitoring and modifying values
- Using a force table to force values in the CPU

To establish an online connection to a configured CPU, click the CPU from the Plant Explorer and click the "Go online" button on the main toolbar.



If this is the first time to go online with this CPU, you probably must set interface configuration before establishing an online connection to a CPU found on that interface.

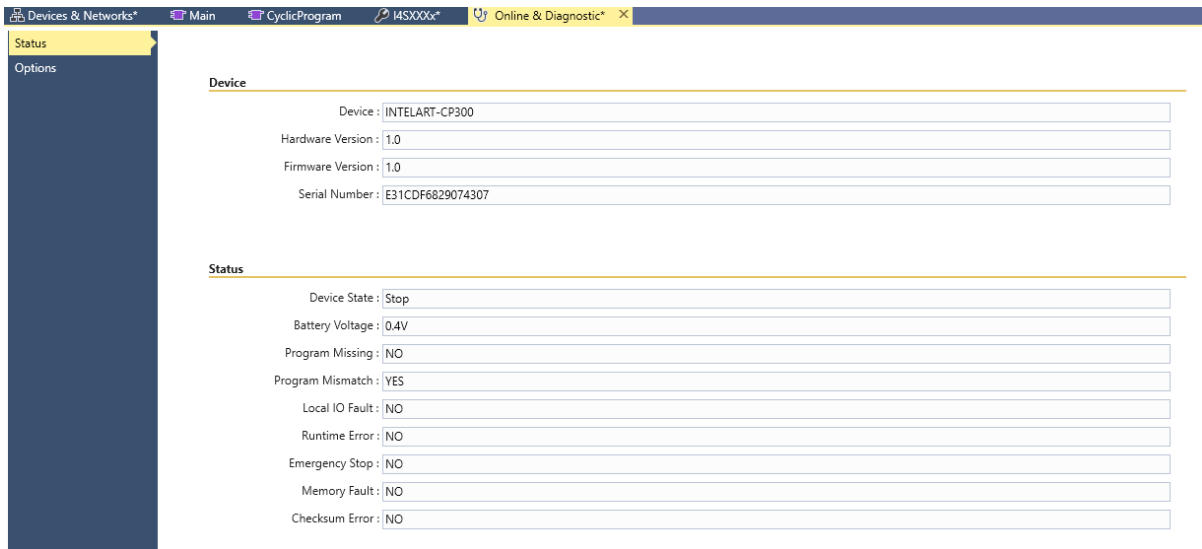


Your programming device is now connected to the CPU. The green color of status bar indicates an online connection. You can now use the Online & diagnostics tools from the Plant Explorer and the Online tools.



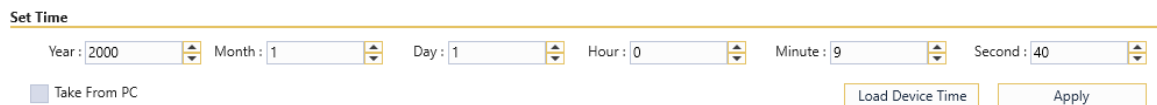
3. Displaying the status of the CPU

You can view the status of an online CPU. Double-click on the "Online & diagnostic" in the Plant Explorer pane then, go to Status tab in the opened editor.



4. Setting the date and time of day

You can set the date and time of day in the online CPU. After accessing "Online & diagnostics" from the Plant Explorer for an online CPU, you can display or set the time and date parameters of the online CPU in the Options tab.



5. Displaying or setting CPU configuration

You can display or set the configuration in the online CPU. After accessing "Online & diagnostics" from the Plant Explorer for an online CPU, go to the Options tab.



NOTICE

Changing in CPU configuration will be taken effect only after restarting the CPU.

6. Resetting to factory settings

You can reset an I4PLC to its original factory settings under the following conditions:

- The CPU is in STOP mode

- Your programming device is disconnected from the CPU
- The connection configuration is set correctly (you can go online to the CPU)

NOTICE

If the CPU is in RUN mode and you start the reset operation, you will get an error. You must place it in STOP mode by RUN/STOP switch.

6.1 Procedure

To reset a CPU to its factory settings, follow these steps:

- 1- Open the Online and Diagnostics view of the CPU.
- 2- Make sure you are offline to the CPU.
- 3- Click the "Factory Reset" button.
- 4- Acknowledge the confirmation prompt with "Yes".



6.2 Result

The CPU is reset to the factory settings:

- The load memory and all operand areas are cleared.
- All parameters are reset to their defaults.

7. CPU operator toolbar for the online CPU

The "CPU operator toolbar" displays the operating mode (STOP or RUN) of the online CPU. The status bar also shows information about CPU and whether the CPU has an error or is in emergency stop.

Use the CPU operating toolbar of the Online Tools to change the operating mode of an online CPU. The Online toolbar is accessible whenever the CPU is online.

8. Monitoring and modifying values in the CPU

Intelart Studio provides online tools for monitoring the CPU:

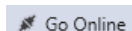
- You can display or monitor the current values of the tags. The monitoring function does not change the program sequence. It presents you with information about the program sequence and the data of the program in the CPU.
- You can also use other functions to control the sequence and the data of the user program:
- You can modify the value for the tags in the online CPU to see how the user program responds.
- You can force a peripheral output (such as Q0.1 or "Start") to a specific value.
- You cannot enable outputs in STOP mode.

WARNING

Always exercise caution when using control functions. These functions can seriously influence the execution of the user/system program.

8.1 Going online to monitor the values in the CPU

To monitor the tags, you must have an online connection to the CPU. Simply click the "Go Online" button in the toolbar.



When you have connected to the CPU, Intelart Studio turns the status bar green.

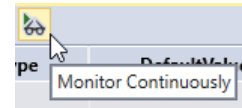


To monitor the execution of the user program and to display the values of the tags, go to "Watch & Force List" in Plant Explorer then, click the "Watch Continuously" button in the toolbar.

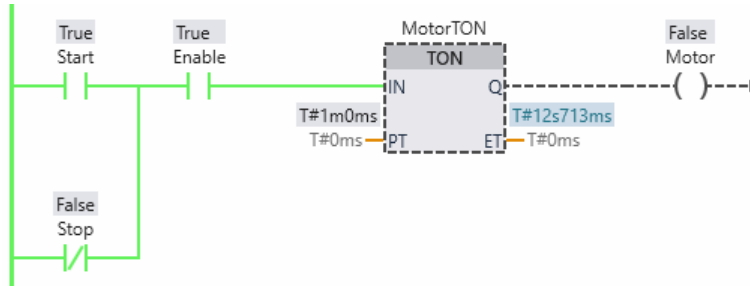


8.2 Displaying status in the program editor

You can monitor the status of the tags in the LAD and FBD program editors. Use the editor bar to display the LAD editor. The editor bar allows you to change the view between the open editors without having to open or close the editors.

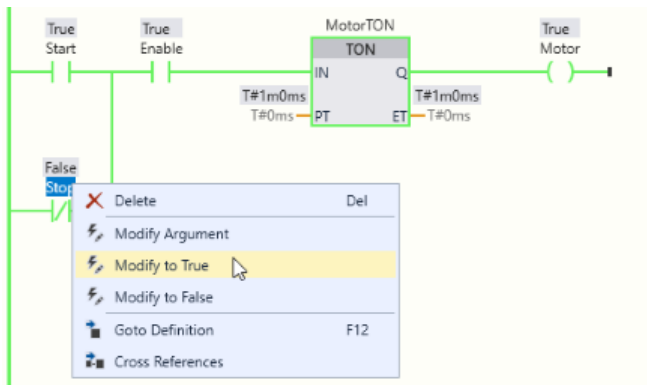


In the toolbar of the program editor, click the "Monitor Continuously" button to display the status of your user program.



The network in the program editor displays power flow in green.

You can also right-click on the instruction or parameter to modify the value for the instruction.



TIP

When you enable a watchlist monitoring or a program block monitoring, Intelart Studio turns the status bar orange.



8.3 Using a watch table to monitor and modify values in the CPU

A watch table allows you to perform monitoring and control functions on data points as the CPU executes your program. These data points can be process image (I or Q), M or G on the monitor or control function. You cannot accurately monitor the physical outputs (Q) because the monitor function can only display the last value written from Q memory and does not read the actual value from the physical outputs. The monitoring function does not change the program sequence. It presents you with information about the program sequence and the data of the program in the CPU.

Control functions enable the user to control the sequence and the data of the program.

Caution must be exercised when using control functions. These functions can seriously influence the execution of the user/system program. The two control functions are Modify and Force.

With the watch table, you can perform the following online functions:

- Monitoring the status of the tags
- Modifying values for the individual tags
- Forcing values in the CPU

- Logging tags data in a csv file on the programming device

Add new watchlist item by clicking on the "Add New Item" button in the toolbar.



Enter the tag name to monitor and select a display format from the dropdown selection. With an online connection to the CPU, clicking the "Watch Continuously" button displays the actual value of the data point in the "Monitor value" field.

The "Modify Selected Items" button modifies the selected tag values by the value provided by "Modify Value" cell.



The "Start Force Selected Items" provides a force function that overwrites the value for a memory or output point to a specified value for the memory tags or peripheral output address. The CPU applies this forced value to the memory tags on demand and output process image before the outputs are written to the modules.

You cannot force an input ("I" address).

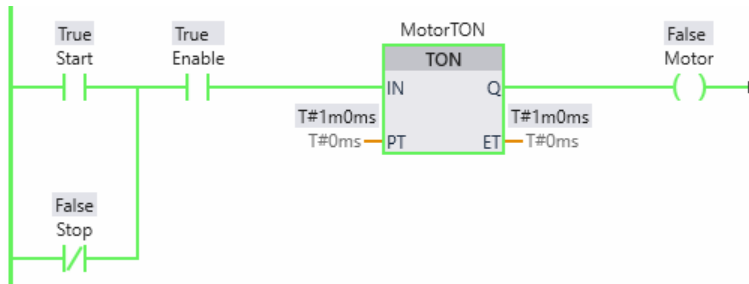
In the "Force value" cell, enter the value for the input or output to be forced. You can then use the check box in the "Select" column to enable forcing of the memory or output.

Use the "Start Force Selected Items" button to force the value of the tags in the table. Click the "Stop Force Selected Items" button to reset the value of the selected forced tags.



Name	Address	Display Format	Monitor Value	Modify Value	Captured Value	Select	Log	Comment	Tag Comment
Motor	...	Default	False	False		<input checked="" type="checkbox"/>	<input type="checkbox"/>		

In the table, you can monitor the status of the forced value for a tag. You can also view the status of the forced value in the program editor.



NOTICE

When a tag is forced, the force actions become part of the current executing program. If you close Intelart Studio, the forced elements remain active in the CPU program until they are cleared or a CPU STOP. To clear these forced elements, you must use Intelart Studio to connect with the online CPU and then use the watchlist to turn off or stop the force function for those elements.

In the program, reads of physical inputs overwrites the forced value. The program uses the forced value in processing. When the program writes a physical output, the output value is overwritten by the force value. The forced value appears at the physical output and is used by the process.

When a tag or output is forced in the watchlist, the force actions become part of the current executing program. Even though the programming software has been closed, the force selections remain active in the operating CPU program until they are cleared by going online with the programming software and stopping the force function. A CPU STOP will clear all the forces states also.

9. Recovery from a lost password

If you have lost the password for a password-protected CPU, you must use factory reset tool (6Resetting to factory settings).

10. Runtime Exceptions

Runtime is a stage of the programming lifecycle. It is the time that a program is running alongside all the external instructions needed for proper execution. Some of these external instructions are called runtime systems or

runtime environments and come as integral parts of the CPU. A runtime system creates a layer over the operating system (OS) that contains other programs that handle the tasks needed to get the main program running. These other programs handle tasks such as allocating memory for the main program and scheduling it.

When a program is at the runtime stage, the executable data of the program is loaded into Application Memory, along with any data that the program references. These may include code that the user did not write but that works in the background to make the program run. It then makes the hardware run the program.

Many users first encounter the term runtime in the context of a runtime exception (runtime error). This refers to a problem with the program that keeps it from executing at runtime due to any damaged, missing or incompatible components or program.

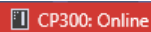

Runtime exceptions can happen for many reasons. The following describes a list of CPU runtime exceptions:

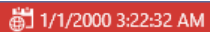
Table 13-1 CPU runtime exceptions

Code	Exception	Description
R000	Unknown Exception	An unknown exception has occurred and the source of this error cannot be identified. Please call the INTELART support.
R001	Overflow Exception	An overflow exception is when the tag type used to store data was not large enough to hold the data. Some tag types can only store numbers up to a certain size. An overflow exception will be produced, for example, if a tag type is SINT and the data to be stored is greater than 127. Also, an invalid cast exception is considered as an overflow exception.
R002	Invalid Type Exception	The Invalid Type Exception represents an error when an operation could not be performed, typically (but not exclusively) when a value is not of the expected type. An Invalid Type Exception may be thrown when: an operand or argument passed to a function or an instruction is incompatible with the type expected by that operator or function. For example, using a VAR_MOVE to pass a DATE to an INT tag will produce an invalid type exception. A null reference exception also is considered as an invalid type exception.
R003	Invalid Name Exception	Invalid Name Exception is a kind of error that occurs when executing a function, tag or user data type that have been used in the code without any previous Declaration. When the CPU cannot identify the global or a local name, it produces an Invalid Name Exception.
R004	OS Exception	OS Exception is a built-in exception in CPU which is raised when an OS specific system function returns a system-related error, including I/O failures such as "file not found" or "memory failure".
R005	Out of Memory Exception	An Out of Memory Exception is raised when an operation fills all the available memory in the CPU. One of the most obvious reasons causing this issue is the complexity of functions (function blocks) call tree or lots of programming instances.
R006	Invalid Value Exception	An Invalid Value Exception is raised when a user gives an invalid value to a function but is of a valid argument. It usually occurs in operations that will require a certain kind of value, even when the value is the correct argument.
R007	Index Out of Range Exception	Index Out of Range Exception is an error that occurs when we try to access an element from an array from an index that is not present in the array. For example, in an array of 10 elements, the index is in the range 0 to 9. If a try to access an element at index 10 or 11 or more, it will cause the CPU to produce an Index Out of Range Exception.
R008	Invalid Program Exception	Invalid Program Exception is an error that occurs when we try to access an element from another element such as a function block or a user data type. For example, in a function block contains 2 elements 'Tag1' and 'Tag2', If a try to access an element with name 'Tag3', it will cause the CPU to produce an Invalid Program Exception.

 TIP

When the CPU




enters a runtime exception or an emergency stop state, Intelart Studio turns the status bar red. By clicking on the “More Detail” button, the Intelart Studio will navigate you to the source of exception.

11. CPU registers

A CPU register is one of a small set of data holding places that are part of the CPU management system. CPU registers is predefined in another memory area called “Special Memory” (S). You can access this area of memory like other memory areas and use tags predefined in this area.

 TIP

An external tag table named “CPU_Registers” generates by Intelart Studio automatically when you add a new device to your project.

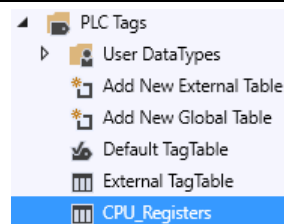


Table 13-2 CPU registers list

Name	Data type	Address	Description
spDevice_ID	UInt	%SW0	Device identification
spHW_VER	UInt	%SW2	Hardware version
spFW_VER	UInt	%SW4	Software version
spSERIAL_NUMBER	ULInt	%S8.0	Device serial number
spTIMESTAMP	DateTime	%SD16	Current timestamp
spUPTIME	Time	%SD20	Device uptime
spBAT_VOLTAGE	UInt	%SW24	Backup battery voltage
spYEAR	USInt	%SB26	Year component of the current date
spMONTH	USInt	%SB27	Month component of the current date
spDAY	USInt	%SB28	Day component of the current date
spHOUR	USInt	%SB29	Hour component of the current date
spMINUTE	USInt	%SB30	Minute component of the current date
spSECOND	USInt	%SB31	Second component of the current date
spWEEKDAY	USInt	%SB32	Weekday component of the current date
spOVERFLOW	Bool	%S40.0	Overflow occurred
spDIV_BY_ZO	Bool	%S40.1	Divide by zero occurred
spIO_ERR	Bool	%S40.2	I/O error state
spTYPE_ERR	Bool	%S40.3	Type error state
spFLASH_STT	Bool	%S40.4	status of load memory
spCOLD_STRT	Bool	%S41.0	Cold start mode
spEMG_STOP	Bool	%S41.1	Emergency stop state
spRUN_MOD	Bool	%S41.2	CPU run mode

Remarks Form

Your comments and recommendations will help us to improve the quality and usefulness of our publications. Please take the first available opportunity to fill out this questionnaire and send it to INTELART.

Please give each of the following questions your own personal mark within a range from 1 (very good) to 5 (very poor).

- 1. Do the contents meet your requirements?
- 2. Is the information you need easy to find?
- 3. Is the text easy to understand?
- 4. Does the level of technical detail meet your requirements?
- 5. Please rate the quality of the graphics and tables.

Additional comments:

NOTICE
Contents of this publication may change without prior notice.
